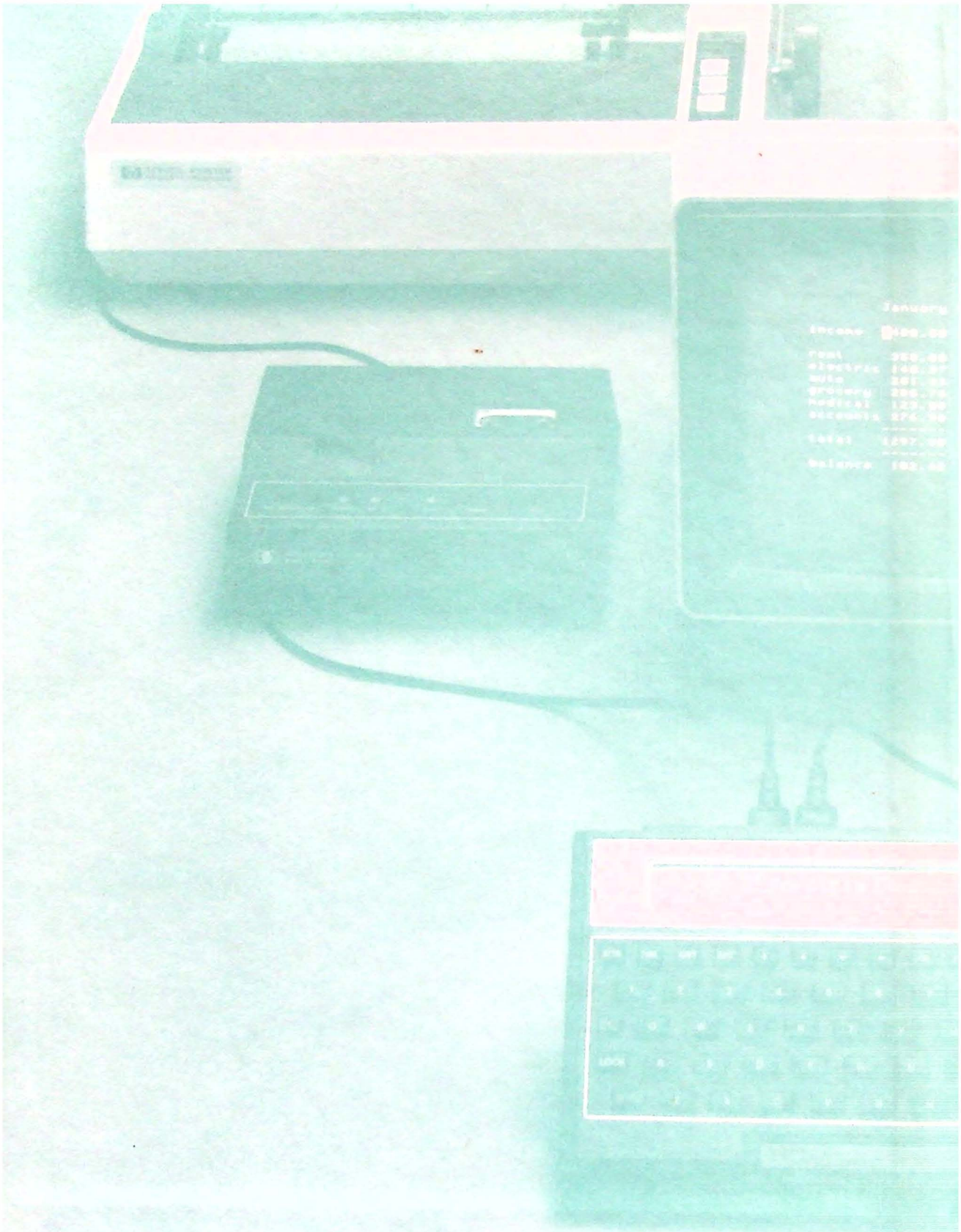
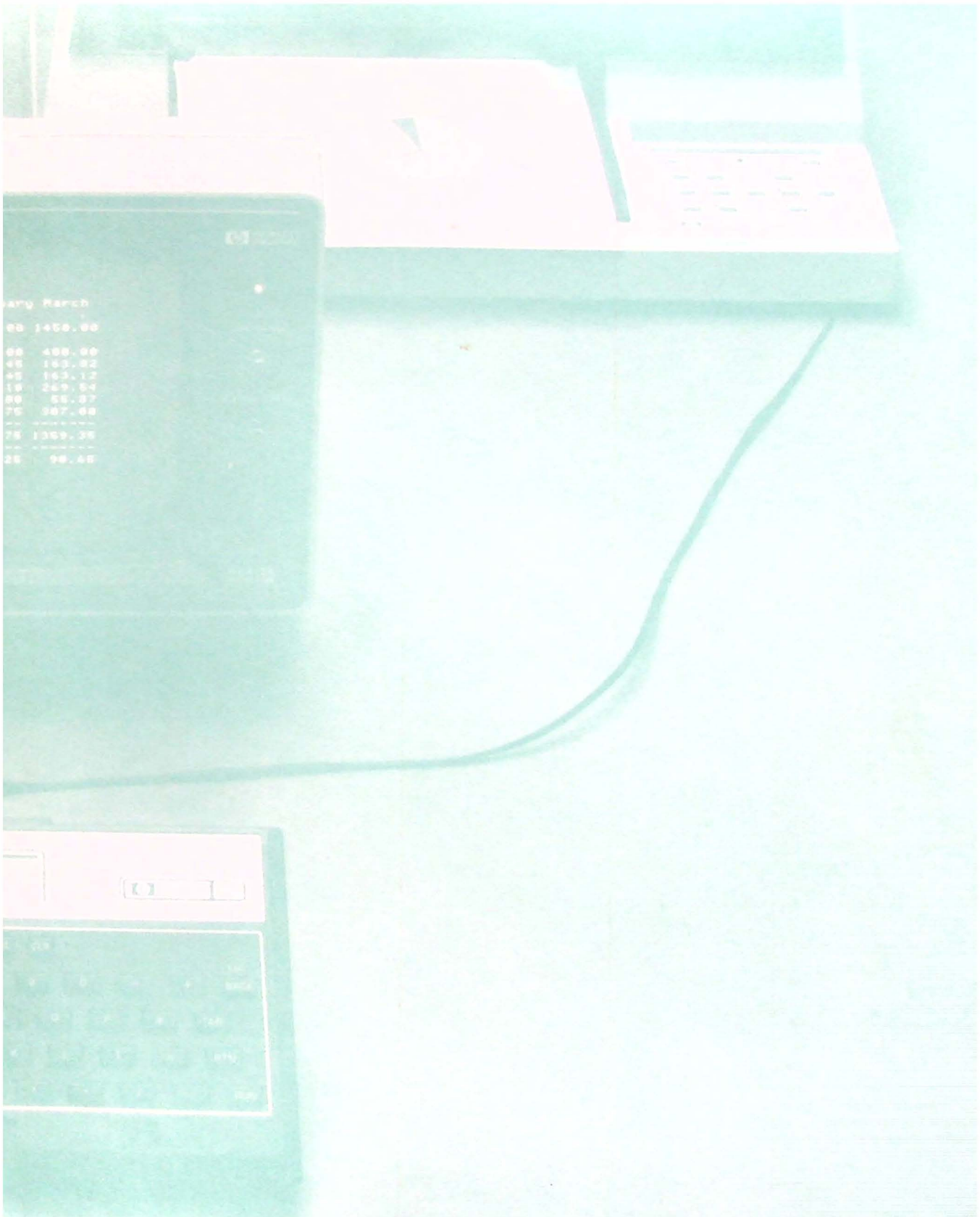
Feichtinger

Arbeitsbuch Mikrocomputer


January
Income
Rent
Electric
Gas
Grocery
Medical
Accounts
Total
Balance



Feichtinger, Arbeitsbuch Mikrocomputer

In der Reihe
Franzis Elektronik-Nachschlagewerk
sind erschienen:

Feichtinger, Mikrocomputer von A—Z
Freyer, Elektronik-Abkürzungen von A—Z
Nührmann, Das große Werkbuch Elektronik
Nührmann, Das kleine Werkbuch Elektronik
Pietsch, Amateurfunk-Lexikon
Renztsch, Begriffe der Elektronik
Rieländer, ABC der Elektroakustik
Rose, Große Elektronik-Formelsammlung

Franzis Elektronik-Nachschlagewerk

Arbeitsbuch Mikrocomputer

Herwig Feichtinger

Funktion und Anwendung von Mikrocomputern, Peripherie und Software

Mit 350 Abbildungen

2., neubearbeitete und erweiterte Auflage

Franzis'

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Feichtinger, Herwig:
Arbeitsbuch Mikrocomputer : Funktion u. Anwendung von Mikrocomputern, Peripherie u. Software / Herwig Feichtinger. — 2., neubearb. u. erw. Aufl. — München : Franzis, 1986.

(Franzis Elektronik-Nachschlagewerk)
ISBN 3-7723-8022-0

© 1987 Franzis-Verlag GmbH, München
© 1987 Elektronik-Verlag Luzern AG.

Sämtliche Rechte, besonders das Übersetzungsrecht, an Text und Bildern vorbehalten.
Fotomechanische Vervielfältigungen nur mit Genehmigung des Verlages.
Jeder Nachdruck - auch auszugsweise - und jegliche Wiedergabe der Bilder sind verboten.

Satz: Rudolf Schaber GmbH, A-4600 Wels
Druck: Wiener Verlag, A-2325 Himberg

Printed in Austria · Imprimé en Autriche

ISB N 3-7723-8022-0

Vorwort

Die Mikrocomputertechnik ist inzwischen derart umfangreich geworden, daß kaum jemand alle Details, die er im Laufe seines Daseins für seine Arbeit benötigt, im Kopf haben kann. Man muß also irgend etwas haben, wo man im Zweifelsfall nachschlagen kann.

In der Tat gibt es heute derart viel Literatur über Mikrocomputer-, Hard- und Software, daß es für wirklich alle Teilgebiete detaillierte Zeitschriften-Beiträge und Bücher gibt. Da die meisten allumfassenden Werke, die sich nicht auf ein Teilgebiet beschränken, sich in erster Linie als oberflächliche Orientierungshilfe an Einsteiger wenden, blieb bisher dem Profi und ernsthaft Interessierten nur übrig, sich erst einmal eine Datenbank (am besten gleich mit Hilfe eines Computers!) anzulegen, in welchem Buch oder welcher Zeitschriften-Ausgabe er zu bestimmten Detailproblemen etwas findet.

Auch das vorliegende Buch ist weit davon entfernt, jedwede Art von Spezialliteratur gänzlich überflüssig zu machen. Denn eine absolute Vollständigkeit wäre bei vertretbarem Umfang und Preis nicht zu erreichen. Aber es stellt für den Praktiker eine erhebliche Arbeitserleichterung dar, wenn er wenigstens für 50% seiner Fragen (oder mehr?) nur noch ein einziges Buch zu Rate ziehen muß und das Sachverzeichnis in dessen hinterem Teil als Datenbank-Ersatz verwenden kann.

Maßgebend dafür, was wirklich aufgenommen wurde und was nicht, war in erster Linie, was man in der täglichen Praxis am meisten braucht: Anschlußbelegungen häufig eingesetzter Bauelemente. Befehlssätze von Mikroprozessoren, Betriebssystemen und höheren Programmiersprachen. Normen für Schnittstellen und Datenübertragungs-Protokolle. Speicherbelegung verbreiteter Computer. Bedienung von Assemblern und Compilern. Und, und, und ...

Aber es wurde auch darauf geachtet, daß dieses Buch nicht zu einem bunten Konglomerat unverständlicher Tabellen und Daten wurde. Überall finden sich Erläuterungen, Hinweise zum Wie und Warum. Deshalb ist es nicht nur ein Nachschlagewerk, sondern auch ein umfassendes Lehrbuch der Mikrocomputertechnik — von einfacher digitaler Logik über den internen Aufbau von Mikroprozessoren bis hin zu komplexen Betriebssystemen wie MS-DOS und Unix.

Den Computer-Herstellern sei insbesondere der letzte Abschnitt des Kapitels 5.1.1 ans Herz gelegt. Gleichzeitig möchte es der Autor nicht versäumen, gerade diesen Herstellern für ihre unvollständigen Handbücher zu danken, sonst wäre nämlich so manches Kapitel in diesem Buch überflüssig gewesen.

Verlag und Autor sind selbstverständlich jederzeit für Verbesserungsvorschläge sowie für Mitteilungen über festgestellte Dreckfuhler dieser Auflage dankbar, damit bei der nächsten die umgekippten Bits wieder gerade gebogen werden können. Denn dieses Buch wurde nahezu vollständig mit einem Computer erstellt, und das schließt (wie schon jeder Laie weiß) eine vollständige Fehlerfreiheit prinzipiell aus.

Schließlich möchte ich den Firmen, die, nichts Böses ahnend, Bildmaterial und Datenblätter zur Verfügung stellten, und meiner Frau Irmtraud für ihren geduldigen Umgang mit dem Textverarbeitungs-Programm danken.

München — Herwig Feichtinger

Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Schaltungen und Verfahren werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden*).
Alle Schaltungen und technischen Angaben in diesem Buch wurden vom Autor mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. Der Verlag und der Autor sehen sich deshalb gezwungen, darauf hinzuweisen, daß sie weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernehmen können. Für die Mitteilung eventueller Fehler sind Autor und Verlag jederzeit dankbar.

*) Bei gewerblicher Nutzung ist vorher die Genehmigung des möglichen Lizenzinhabers einzuholen.

Inhalt

1 Hardware eines Mikrocomputers ... 11

1.1 Grundlagen ... 11
1.1.1 Funktionsweise eines Mikrocomputers ... 11
1.1.2 Duale und hexadezimale Zahlen ... 15
1.1.3 Zeichendarstellung in Computern ... 21

1.2 Mikroprozessoren ... 25
1.2.1 Überblick ... 25
1.2.2 Registerstruktur ... 36
1.2.3 Gehäuseformen und Anschlußbelegung ... 46
1.2.4 Elektrische Eigenschaften ... 55
1.2.5 Befehlssätze einiger Mikroprozessoren ... 66

1.3 Halbleiterspeicher ... 90
1.3.1 Statische Speicher ... 90
1.3.2 Dynamische Speicher ... 91
1.3.3 Festwertspeicher ... 95

1.4 Peripherie-Bausteine ... 100
1.4.1 Port- und Timer-Bausteine ... 100
1.4.2 Bausteine für serielle Ein- und Ausgabe ... 112
1.4.3 D/A- und A/D-Umsetzer ... 116
1.4.4 Video-Controller ... 125
1.4.5 Floppy-Disk-Controller ... 132
1.4.6 Sonstige Peripheriebausteine ... 135

1.5 Unvermeidbarer »Kleinkram« ... 138
1.5.1 Diskrete Bauelemente ... 138
1.5.2 TTL-Schaltungen ... 151
1.5.3 CMOS-Schaltungen ... 158
1.5.4 Stromversorgungs-Schaltungen ... 162
1.5.5 Sensoren ... 168

1.6 Aufbautechniken ... 173
1.6.1 Fädel- und Wire-Wrap-Verdrahtung ... 173
1.6.2 Gedruckte Schaltungen ... 175
1.6.3 Busverdrahtung ... 179
1.6.4 Gehäuse, Störstrahlung, Wärmeabfuhr ... 189

1.7 Verbindung nach draußen ... 191
1.7.1 Tastaturen ... 191
1.7.2 Alphanumerische Anzeigen ... 195
1.7.3 Schnittstellen ... 201

1.8 Externe Speicherverfahren ... 216
1.8.1 Floppy-Disk-Aufzeichnung ... 216
1.8.2 Festplatten (Hard Disks) ... 227
1.8.3 Kassettenaufzeichnung ... 230
1.8.4 Der Strichcode ... 235
1.8.5 Lochstreifen-Datenformate ... 240

1.9 Schaltungstechnik bei Mikrocomputern ... 243
1.9.1 Allgemeines ... 243
1.9.2 Ein Z80-System ... 248
1.9.3 Ein 6504-Einplatinen-Computer ... 251

2 Peripheriegeräte ... 254

2.1 Ausgabegeräte ... 254
2.1.1 Matrixdrucker ... 254
2.1.2 Typenrad- und Typenkorb-Drucker ... 261
2.1.3 Plotter ... 264
2.1.4 Sprachausgabe ... 268

2.2 Kommunikations-Einrichtungen ... 273
2.2.1 Modems ... 273
2.2.2 Akustikkoppler ... 278
2.2.3 Der Datex-P-Dienst ... 280
2.2.4 Datenbanken und Mailbox-Dienste ... 285
2.2.5 Lokale Netzwerke ... 293

2.3 Spezielle Eingabeverfahren ... 296
2.3.1 Die „Maus" ... 296
2.3.2 Joysticks ... 300
2.3.3 Grafik-Tablets und Lichtgriffel ... 301
2.3.4 Spracherkennung ... 303

3 Programmierung von Mikrocomputern ... 307

3.1 Maschinensprache ... 307
3.1.1 Das Monitorprogramm ... 307
3.1.2 Hexadezimale Programmeingabe ... 309
3.1.3 Assembler ... 323

3.2 Höhere Programmiersprachen ... 344
3.2.1 Interpreter ... 344
3.2.2 Compiler ... 350
3.2.3 Basic ... 352
3.2.4 Forth ... 402
3.2.5 Pascal ... 415
3.2.6 Weitere Sprachen ... 425

3.3 Grafische Darstellung von Programmen, Befehlen und Daten ... 429
3.3.1 Flußdiagramme ... 429
3.3.2 Struktogramme ... 432
3.3.3 Syntax-Diagramme ... 435
3.3.4 Datenstrukturen ... 437

4 Mikrocomputer-Betriebssysteme 444

4.1 Standard-Betriebssysteme 444
4.1.1 CP/M 445
4.1.2 MS-DOS 468
4.1.3 Unix 501
4.1.4 Sonstige Standard-Betriebssysteme 514

4.2 Herstellerspezifische Betriebssysteme 517
4.2.1 Überblick 517
4.2.2 Apple-DOS 3.3 519
4.2.3 Commodore-DOS 532

5 Anwender-Software 540

5.1 Anforderungen an Anwender-Software 540
5.1.1 Ergonomie 540
5.1.2 Kosten und Portabilität 546
5.1.3 Schutz vor Kopierern 550
5.1.4 Paßwortschutz 556

5.2 Entwicklung, Debugging, Marketing 561
5.2.1 Entwicklungsphasen 561
5.2.2 Marketing 565
5.2.3 Veröffentlichung 569

5.3 Kommerzielle Anwenderprogramme 570
5.3.1 Wordstar 570
5.3.2 Applewriter-II 574

5.4 Algorithmen 576
5.4.1 Funktionen 576
5.4.2 Numerische Algorithmen 578
5.4.3 String-Routinen 585
5.4.4 Sortierverfahren 587
5.4.5 Grafik-Algorithmen 590

6 Adressen 597

6.1 Hersteller von Computern und Peripherie 597
6.2 Hersteller von Bauelementen und Baugruppen 602
6.3 Hersteller von Akustikkopplern 605
6.4 Software-Hersteller 605
6.5 Institutionen und Verbände 606

7 Quellenverzeichnis 608

Sachverzeichnis 610

1 Hardware eines Mikrocomputers

1.1 Grundlagen

1.1.1 Funktionsweise eines Mikrocomputers

Das Wort „Mikrocomputer" setzt sich zusammen aus „Mikro" (sehr klein) und „Computer" (engl. to compute = rechnen). Ein sehr kleiner Rechner also? Nein, kein Taschenrechner. Der Mikrocomputer kann viel mehr als nur mit Ziffern rechnen. Er kann an irgendwelchen Anschlüssen zum Beispiel Daten von einer Schreibmaschinen-Tastatur erhalten, die getippten Buchstaben speichern, später Korrekturen zulassen und schließlich den ganzen Text an einen angeschlossenen Drucker ausgeben. Das Manuskript zu diesem Buch ist genau auf diese Weise entstanden — auf einem Computer namens Apple-II. Sehen wir uns seinen Aufbau einmal näher an *(Abb. 1.1.1.1)*.

Die 51 Tasten des Apple-II sind natürlich nicht direkt mit dem Mikroprozessor, dem Herz des Systems, verbunden, denn der hat ja nur insgesamt 40 Anschlußbeine. Vielmehr ist ein Tastatur-Encoder vorhanden, der vom Prozessor regelmäßig über einen Adressenbus angesprochen wird und der die Information, ob und welche Taste gedrückt wurde, auf nur acht Leitungen über den Datenbus zum Prozessor (hier ein „6502") sendet. Als Bus bezeichnet man in der Computertechnik einfach eine Gruppe von Leitungen — wie der Omnibus ebenfalls ein Transportmittel, hier natürlich für elektrische Signale.

Je nachdem, welche Kombination von „Spannung" und „keine Spannung" auf den Leitungen eines Busses anliegt, können sehr unterschiedliche Informationen auf dem Bus übertragen werden. Über den Adressenbus wählt der Prozessor (die CPU = Central Processing Unit) aus, zu oder von welchem anderen Baustein des Systems anschließend ein Datentransfer über den Datenbus stattfinden kann.

Wählt der Mikroprozessor die Adresse des Tasten-Encoders an, so legt letzterer den Code der zuletzt gedrückten Taste als Spannungskombination auf die acht Leitungen des Datenbusses (weil er acht Leitungen umfaßt, bezeichnet man den verwendeten Prozessor 6502 als 8-Bit-CPU).

Bestimmte Adressenbereiche sind auch den Speicherzellen von ROM und RAM zugeordnet: ROM bedeutet Festwertspeicher (Read Only Memory), der bei der Herstellung fest programmiert wurde und aus dem der Prozessor nur lesen kann. In das RAM (Random Access Memory, Speicher mit wahlfreiem Zugriff bzw. Schreib-Lese-Speicher) kann die CPU dagegen auch Informationen hineinschrei-

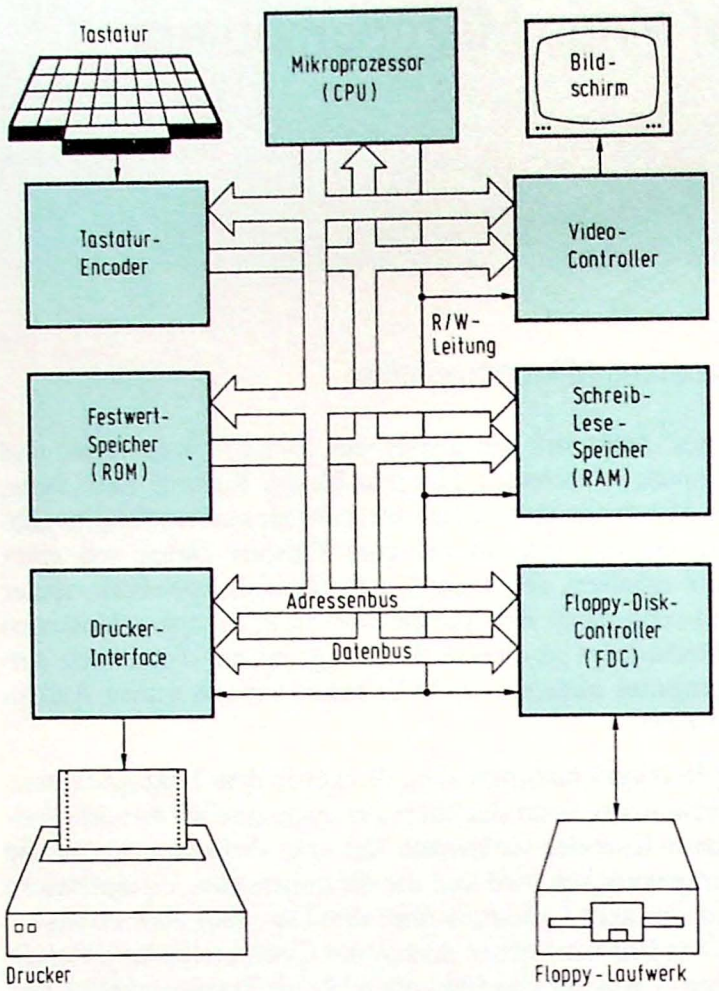


Abb. 1.1.1.1
Aufbau eines typischen Tischcomputers. Die einzelnen Bausteine werden vom Mikroprozessor über den Adressenbus adressiert, der Datentransfer erfolgt über den Datenbus

ben, indem sie diesem Speicher über eine besondere Steuerleitung (Read/Write = Lesen/Schreiben) mitteilt, ob aus ihm gelesen oder in ihn geschrieben werden soll. Allerdings geht der RAM-Inhalt verloren, wenn man den Computer vorübergehend ausschaltet.

Die R/W-Leitung gehört zu einer dritten Gruppe von Leitungen, dem Steuerbus. Er dient z.B. auch zur Weiterleitung des Taktsignals (hier rund 1 MHz, also eine Million Spannungsimpulse pro Sekunde), das den zeitlichen Ablauf innerhalb des Computers regelt.

Was aber passiert beim Einschalten des Computers? Woher weiß der Prozessor, was zu tun ist? Nun, dafür gibt es den sogenannten Reset-Eingang der CPU (Rücksetz-Eingang). Eine spezielle Schaltung (ein Monoflop) erzeugt an ihm kurz nach dem Einschalten einen Impuls, und der Prozessor weiß jetzt, daß er an einer bestimmten Speicherzelle mit der Abarbeitung des Programms beginnen soll. Die Adresse dieser Speicherzelle liegt im ROM.

Das dort stehende Programm (es ist in der für den Prozessor direkt verständlichen Maschinensprache geschrieben) löscht unter anderem erst einmal den angeschlossenen Bildschirm sowie den Schreib-Lese-Speicher (dessen Inhalt ja noch undefiniert ist) und fragt dann die Tastatur ab. Über sie kann man nun z.B. ein Kommando eingeben, das von einem angeschlossenen Floppy-Disk-Laufwerk ein Textverarbeitungs-Programm ins RAM lädt und startet. Dieses Programm wiederum speichert die eingetippten Buchstaben eines Manuskripts an eine noch freie Stelle des RAM, von wo sie später ebenfalls auf Floppy abgespeichert oder auf einen Drucker ausgegeben werden können.

Selbstverständlich sind auch den anderen Systemkomponenten wie Floppy-Disk- und Video-Interface (Anpaßschaltung) bestimmte Adressen zugeordnet, so daß stets ein eindeutiger Zugriff von der CPU her möglich ist. Konkret ist der zeitliche Ablauf beim Schreiben einer Information in eine RAM-Zelle wie folgt:

1. Der Prozessor legt die Adresse der gewünschten RAM-Zelle auf den Adressenbus und teilt dem Speicher über die Read/Write-Leitung mit, daß nun ins RAM geschrieben werden soll.
2. Nun legt die CPU die zu schreibenden Daten auf den Datenbus, so daß sie vom RAM übernommen werden können.
3. Der Prozessor stellt die R/W-Leitung auf „Lesen" zurück und legt die Adresse des nächsten Programmbefehls auf den Adressenbus.
4. Der angesprochene Speicherbaustein (ROM oder RAM) übergibt der CPU den nächsten Befehl auf dem Datenbus.

Die zeitliche Abfolge dieser Vorgänge wird vom Taktgenerator gesteuert. Er ist meist mit auf dem Silizium-Chip der CPU integriert und erfordert im wesentlichen extern nur einen Schwingquarz als frequenzbestimmendes Bauelement.

Fassen wir zusammen:

1. Das Herz des Mikrocomputers ist der Mikroprozessor, auch CPU genannt. Die CPU wählt die übrigen Systemkomponenten über den Adressenbus an.
2. Anwenderprogramme und -daten werden „flüchtig" im RAM gespeichert, sind also beim Ausschalten verloren, wenn sie vorher nicht auf einen externen Datenträger (Floppy-Disk oder Kassette) abgespeichert werden.
3. Da beim Einschalten der RAM-Inhalt noch undefiniert ist, muß ein bestimmtes Minimal-Programm im ROM stehen, das z.B. das Laden von Anwenderprogrammen erlaubt.

Soviel einstweilen zur Theorie. Wie ein Mikrocomputer in der Praxis aussieht, hängt in erster Linie von seinem Verwendungszweck ab. Grob kann man zwischen Einplatinen- und Tischcomputern unterscheiden.

Einplatinen-Computer (engl. Single Board Computer, SBC) dienen vornehmlich Steuerungszwecken. Sie besitzen das nötige Programm vollständig in einem Festwertspeicher, werden also nur einmal für eine bestimmte Aufgabe programmiert.

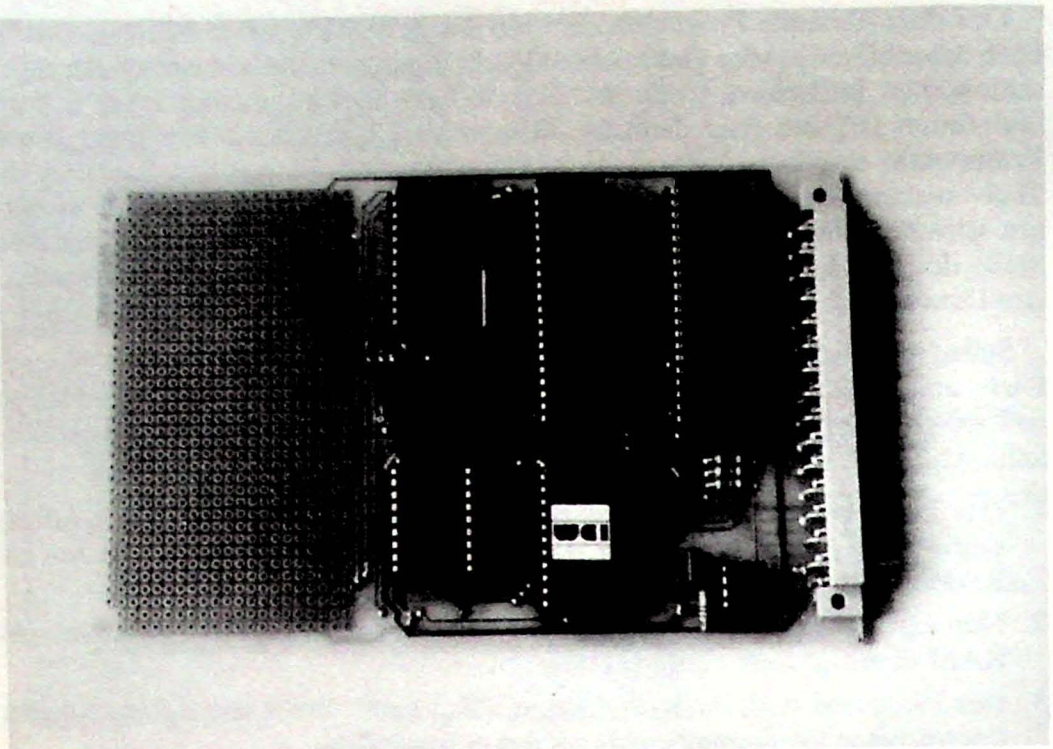


Abb. 1.1.1.2 Einplatinen-Computer dienen in erster Linie für Steuerungszwecke und sind für eine bestimmte Aufgabe fest programmiert

Ein Umprogrammieren ist nur durch Austauschen des Festwertspeichers möglich. Über Ein-/Ausgabeleitungen können sie z. B. Temperaturfühler oder Drucksensoren abfragen und Motoren oder Ventile steuern. Typische Anwendungen sind Fahrkarten-Automaten, Heizungssteuerungen, Verbrauchsoptimierung in Verbrennungsmotoren, Spielautomaten und Aufzugssteuerungen. Einplatinencomputer sind oft nicht größer als eine Zigarettenpackung *(Abb. 1.1.1.2)*.

Es ist sogar möglich, alle wichtigen Bestandteile eines Mikrocomputers auf einem einzigen Silizium-Chip zu integrieren, d.h. CPU, RAM, ROM und Ein-/Ausgabe. Dann spricht man von Einchip-Mikrocomputern. Auch sie werden vorwiegend fest programmiert für Steuerungszwecke eingesetzt.

Tischcomputer erlauben es dagegen dem Benutzer, eigene Programme einzutippen oder von einem externen Speichermedium (z. B. Floppy-Disk) ins RAM zu laden. Zu diesem Zweck besitzen sie eine alphanumerische Tastatur und eine Anzeige (z. B. Bildschirm oder Anschluß für Fernsehgeräte). Ihre Größe ist meist durch die der Tastatur vorgegeben und entspricht daher mindestens etwa der einer Reiseschreibmaschine *(Abb. 1.1.1.3)*.

Unter der Bezeichnung „Peripheriegeräte" faßt man alles zusammen, was nicht im Gehäuse des Tischcomputers selbst untergebracht ist. Das können ein externes

Abb. 1.1.1.3 Tischcomputer müssen nicht notwendigerweise mit einem Bildschirm ausgerüstet sein, eine großflächige LC-Anzeige ist besonders bei tragbaren Geräten auch oft zu finden (PX-8 von Epson)

Floppy-Disk-Laufwerk zur Speicherung von Programmen und Daten, ein Drucker oder ein Telefonmodem zur Übertragung von Daten über das Fernsprechnetz sein. (Den Peripheriegeräten ist ein eigenes Kapitel gewidmet.)

1.1.2 Duale und hexadezimale Zahlen

Wie erwähnt, ist es in einem Mikrocomputer nötig, Informationen mit den zwei elektrischen Zuständen „keine Spannung da" und „Spannung da" zu transferieren. Diese zweiwertige Informationsdarstellung nennt man auch „binäre Logik". In der Praxis ist es schwierig, ideale elektronische Schalter zu realisieren; insbesondere ist es real kaum möglich, eine Spannung ganz abzuschalten. Man relativiert die Aussage daher mit den beiden Zuständen „Low" (geringe Spannung) und „High" (höhere Spannung). Beispielsweise entspricht eine Spannung von 0 ... 0,7 Volt dem Low- und 2,4 ... 5 V dem High-Pegel (siehe TTL-Schaltungen).

High und Low

Bei der meist verwendeten positiven Logik bezeichnet man den High-Pegel mit der Ziffer 1 und den Low-Pegel mit 0. Theoretisch wäre es auch möglich, die Null dem High-Pegel und die Eins dem Low-Pegel zuzuordnen; dann spricht man von negativer Logik. Doch bleiben wir, der üblichen Praxis folgend, bei der positiven Logik.

Mit einer einzigen Leitung, z.B. des Datenbusses eines Computers, kann man nur entweder 0 oder 1 übertragen. Bekanntlich rechnen aber Computer durchaus auch mit größeren Zahlen als 1. Dafür braucht man mehr Leitungen. So ist es beispielsweise mit zwei Leitungen schon möglich, vier unterschiedliche Zahlen zu codieren:

Leitung 1	Leitung 2	Zahl
Low (0)	Low (0)	„0“
Low (0)	High (1)	„1“
High (1)	Low (0)	„2“
High (1)	High (1)	„3“

Jeder möglichen Kombination von Spannungspegeln auf den Leitungen wird also jeweils eine Zahl zugeordnet. Man kann sich leicht ausrechnen, wie viele Kombinationsmöglichkeiten und damit darstellbare Zahlen (k) bei einer bestimmten Anzahl von Leitungen (n) es gibt:

$k = 2 \uparrow n$

Beginnt man die Zählung stets mit Null, so ist die größte darstellbare Zahl m mit n Leitungen:

$m = 2 \uparrow n - 1$

Die auf einer einzigen Leitung so übertragbare Informationseinheit nennt man 1 Bit. Die Aussage eines Bits kann nur zwischen zwei Alternativen unterscheiden: nein oder ja, 0 oder 1, falsch oder richtig, unwahr oder wahr. Statt der Zahl der Leitungen kann man in die beiden obigen Formeln genauso gut die Zahl der Informationsbits einsetzen: Mit zwei Bits lassen sich vier Kombinationen von Nullen und Einsen, also vier unterschiedliche Zahlenwerte darstellen.

Bei den meisten Mikrocomputern umfaßt der Datenbus acht Leitungen, ist also 8 Bit „breit“. Entsprechend einer achtspurigen Straße, auf der acht Autos nebeneinander fahren können, können hier acht Bits parallel (gleichzeitig) übertragen werden, z.B. vom Prozessor zu einem Speicherbaustein.

Diese acht Datenbits lassen $2 \uparrow 8 = 256$ Kombinationen von Nullen und Einsen zu. Die größte damit darstellbare Zahl ist $2 \uparrow 8 - 1 = 255$.

Dualzahlen

Schreibt man die Nullen und Einsen der acht Leitungen nebeneinander hin, so nennt man das Ergebnis eine „Dualzahl“, da nur zwei unterschiedliche Ziffern ver-

Dualzahl	Dez.	Dualzahl	Dez.	Dualzahl	Dez.	Dualzahl	Dez.	Dualzahl	Dez.	Dualzahl	Dez.
00000000	0	00000001	1	00000010	2	00000011	3	00000100	4	00000101	5
00000110	6	00000111	7	00001000	8	00001001	9	00001010	10	00001011	11
00001100	12	00001101	13	00001110	14	00001111	15	00010000	16	00010001	17
00010010	18	00010011	19	00010100	20	00010101	21	00010110	22	00010111	23
00011000	24	00011001	25	00011010	26	00011011	27	00011100	28	00011101	29
00011110	30	00011111	31	00100000	32	00100001	33	00100010	34	00100011	35
00100100	36	00100101	37	00100110	38	00100111	39	00101000	40	00101001	41
00101010	42	00101011	43	00101100	44	00101101	45	00101110	46	00101111	47
00110000	48	00110001	49	00110010	50	00110011	51	00110100	52	00110101	53
00110110	54	00110111	55	00111000	56	00111001	57	00111010	58	00111011	59
00111100	60	00111101	61	00111110	62	00111111	63	01000000	64	01000001	65
01000010	66	01000011	67	01000100	68	01000101	69	01000110	70	01000111	71
01001000	72	01001001	73	01001010	74	01001011	75	01001100	76	01001101	77
01001110	78	01001111	79	01010000	80	01010001	81	01010010	82	01010011	83
01010100	84	01010101	85	01010110	86	01010111	87	01011000	88	01011001	89
01011010	90	01011011	91	01011100	92	01011101	93	01011110	94	01011111	95
01100000	96	01100001	97	01100010	98	01100011	99	01100100	100	01100101	101
01100110	102	01100111	103	01101000	104	01101001	105	01101010	106	01101011	107
01101100	108	01101101	109	01101110	110	01101111	111	01110000	112	01110001	113
01110010	114	01110011	115	01110100	116	01110101	117	01110110	118	01110111	119
01111000	120	01111001	121	01111010	122	01111011	123	01111100	124	01111101	125
01111110	126	01111111	127	10000000	128	10000001	129	10000010	130	10000011	131
10000100	132	10000101	133	10000110	134	10000111	135	10001000	136	10001001	137
10001010	138	10001011	139	10001100	140	10001101	141	10001110	142	10001111	143
10010000	144	10010001	145	10010010	146	10010011	147	10010100	148	10010101	149
10010110	150	10010111	151	10011000	152	10011001	153	10011010	154	10011011	155
10011100	156	10011101	157	10011110	158	10011111	159	10100000	160	10100001	161
10100010	162	10100011	163	10100100	164	10100101	165	10100110	166	10100111	167
10101000	168	10101001	169	10101010	170	10101011	171	10101100	172	10101101	173
10101110	174	10101111	175	10110000	176	10110001	177	10110010	178	10110011	179
10110100	180	10110101	181	10110110	182	10110111	183	10111000	184	10111001	185
10111010	186	10111011	187	10111100	188	10111101	189	10111110	190	10111111	191
11000000	192	11000001	193	11000010	194	11000011	195	11000100	196	11000101	197
11000110	198	11000111	199	11001000	200	11001001	201	11001010	202	11001011	203
11001100	204	11001101	205	11001110	206	11001111	207	11010000	208	11010001	209
11010010	210	11010011	211	11010100	212	11010101	213	11010110	214	11010111	215
11011000	216	11011001	217	11011010	218	11011011	219	11011100	220	11011101	221
11011110	222	11011111	223	11100000	224	11100001	225	11100010	226	11100011	227
11100100	228	11100101	229	11100110	230	11100111	231	11101000	232	11101001	233
11101010	234	11101011	235	11101100	236	11101101	237	11101110	238	11101111	239
11110000	240	11110001	241	11110010	242	11110011	243	11110100	244	11110101	245
11110110	246	11110111	247	11111000	248	11111001	249	11111010	250	11111011	251
11111100	252	11111101	253	11111110	254	11111111	255				

Abb. 1.1.2.1 Umrechnungstabelle für duale und dezimale Zahlen bis 255

wendet werden. *Abb. 1.1.2.1* enthält eine Übersicht der 256 möglichen Kombinationen. In der Computertechnik hat sich für eine solche Gruppe von 8 Bits die Einheit „1 Byte" eingebürgert. Eine 4-Bit-Gruppe nennt man „Nibble".

Die Umrechnung einer Dualzahl in die entsprechende Dezimalzahl kann über Zweierpotenzen erfolgen. Als Beispiel soll die Dualzahl 10101110 in ihr dezimales Äquivalent umgerechnet werden:

$$1\quad 0\quad 1\quad 0\quad 1\quad 1\quad 1\quad 0\quad =$$
$$1\times 2\uparrow 7+0\times 2\uparrow 6+1\times 2\uparrow 5+0\times 2\uparrow 4+1\times 2\uparrow 3+1\times 2\uparrow 2+1\times 2\uparrow 1+0\times 2\uparrow 0=$$
$$1\times 128+0\times 64\ +1\times 32\ +0\times 16\ +1\times 8\ \ +1\times 4\ \ +1\times 2\ \ +0\times 1\ \ =$$
$$128+32+8+4+2=174$$

Das ganz links hingeschriebene Bit (hier 1) bezeichnet man als höchstwertiges Bit (MSB = Most Significant Bit), weil es bei der Umrechnung mit dem größten Multiplikator eingeht. Das Bit ganz rechts ist das niederwertigste und wird LSB = Least Significant Bit genannt.

Die Umrechnung ist sicher relativ umständlich. Zwar interessiert im Grunde beispielsweise nicht, wie man die duale Adresse einer bestimmten Speicherzelle nun dezimal schreibt. Andererseits wäre es gerade bei dem 16 Bit (16 Leitungen) umfassenden Adressenbus umständlich, jede Adresse als Folge von 16 Einsen und Nullen schreiben zu müssen.

Sedezimale Zahlendarstellung

In der Mikrocomputertechnik hat sich deshalb die sedezimale Schreibweise von Adressen und Daten eingebürgert, landläufig (wenn auch nicht ganz so korrekt) hexadezimal genannt. Ähnlich, wie das duale Zahlensystem nur zwei und das dezimale zehn Ziffern kennt, umfaßt das Sechzehner-System 16 Ziffern. Weil dafür die zehn arabischen Ziffern 0 ... 9 nicht ausreichen, nimmt man noch die sechs Buchstaben A ... F hinzu.

Abb. 1.1.2.2 zeigt die sich ergebende Zuordnung von dualen und hexadezimalen Zahlen. Der Vorteil des Sechzehner-Systems liegt auf der Hand: Jeweils vier Bits der Dualzahl entsprechen einer hexadezimalen Ziffer. Die Umrechnung ist also wesentlich einfacher als bei der Dezimal-Darstellung. Die acht Bits eines Bytes ergeben immer zwei Hex-Ziffern. (Obwohl das Wort „hex" fälschlich „sechs" statt „sechzehn" bedeutet, ist es in der Fachsprache durchaus üblich.) Die 16 Adressenbits eines Mikrocomputers lassen sich mit nur vier Hex-Ziffern darstellen; im Dezimalsystem wären hierfür bis zu fünf Ziffern nötig ($2\uparrow 16 - 1 = 65535$).

Dualzahl	Hex	Dualzahl	Hex	Dualzahl	Hex	Dualzahl	Hex	Dualzahl	Hex	Dualzahl	Hex
00000000	00	00000001	01	00000010	02	00000011	03	00000100	04	00000101	05
00000110	06	00000111	07	00001000	08	00001001	09	00001010	0A	00001011	0B
00001100	0C	00001101	0D	00001110	0E	00001111	0F	00010000	10	00010001	11
00010010	12	00010011	13	00010100	14	00010101	15	00010110	16	00010111	17
00011000	18	00011001	19	00011010	1A	00011011	1B	00011100	1C	00011101	1D
00011110	1E	00011111	1F	00100000	20	00100001	21	00100010	22	00100011	23
00100100	24	00100101	25	00100110	26	00100111	27	00101000	28	00101001	29
00101010	2A	00101011	2B	00101100	2C	00101101	2D	00101110	2E	00101111	2F
00110000	30	00110001	31	00110010	32	00110011	33	00110100	34	00110101	35
00110110	36	00110111	37	00111000	38	00111001	39	00111010	3A	00111011	3B
00111100	3C	00111101	3D	00111110	3E	00111111	3F	01000000	40	01000001	41
01000010	42	01000011	43	01000100	44	01000101	45	01000110	46	01000111	47
01001000	48	01001001	49	01001010	4A	01001011	4B	01001100	4C	01001101	4D
01001110	4E	01001111	4F	01010000	50	01010001	51	01010010	52	01010011	53
01010100	54	01010101	55	01010110	56	01010111	57	01011000	58	01011001	59
01011010	5A	01011011	5B	01011100	5C	01011101	5D	01011110	5E	01011111	5F
01100000	60	01100001	61	01100010	62	01100011	63	01100100	64	01100101	65
01100110	66	01100111	67	01101000	68	01101001	69	01101010	6A	01101011	6B
01101100	6C	01101101	6D	01101110	6E	01101111	6F	01110000	70	01110001	71
01110010	72	01110011	73	01110100	74	01110101	75	01110110	76	01110111	77
01111000	78	01111001	79	01111010	7A	01111011	7B	01111100	7C	01111101	7D
01111110	7E	01111111	7F	10000000	80	10000001	81	10000010	82	10000011	83
10000100	84	10000101	85	10000110	86	10000111	87	10001000	88	10001001	89
10001010	8A	10001011	8B	10001100	8C	10001101	8D	10001110	8E	10001111	8F
10010000	90	10010001	91	10010010	92	10010011	93	10010100	94	10010101	95
10010110	96	10010111	97	10011000	98	10011001	99	10011010	9A	10011011	9B
10011100	9C	10011101	9D	10011110	9E	10011111	9F	10100000	A0	10100001	A1
10100010	A2	10100011	A3	10100100	A4	10100101	A5	10100110	A6	10100111	A7
10101000	A8	10101001	A9	10101010	AA	10101011	AB	10101100	AC	10101101	AD
10101110	AE	10101111	AF	10110000	B0	10110001	B1	10110010	B2	10110011	B3
10110100	B4	10110101	B5	10110110	B6	10110111	B7	10111000	B8	10111001	B9
10111010	BA	10111011	BB	10111100	BC	10111101	BD	10111110	BE	10111111	BF
11000000	C0	11000001	C1	11000010	C2	11000011	C3	11000100	C4	11000101	C5
11000110	C6	11000111	C7	11001000	C8	11001001	C9	11001010	CA	11001011	CB
11001100	CC	11001101	CD	11001110	CE	11001111	CF	11010000	D0	11010001	D1
11010010	D2	11010011	D3	11010100	D4	11010101	D5	11010110	D6	11010111	D7
11011000	D8	11011001	D9	11011010	DA	11011011	DB	11011100	DC	11011101	DD
11011110	DE	11011111	DF	11100000	E0	11100001	E1	11100010	E2	11100011	E3
11100100	E4	11100101	E5	11100110	E6	11100111	E7	11101000	E8	11101001	E9
11101010	EA	11101011	EB	11101100	EC	11101101	ED	11101110	EE	11101111	EF
11110000	F0	11110001	F1	11110010	F2	11110011	F3	11110100	F4	11110101	F5
11110110	F6	11110111	F7	11111000	F8	11111001	F9	11111010	FA	11111011	FB
11111100	FC	11111101	FD	11111110	FE	11111111	FF				

Abb. 1.1.2.2 Umrechnungstabelle für duale und hexadezimale Zahlen. Jede Hex-Ziffer entspricht vier Bits bzw. vier dualen Ziffern

Der Vollständigkeit halber sei noch darauf hingewiesen, daß manche weniger verbreitete Computer auch mit oktaler Zahlendarstellung arbeiten. Hierbei stehen die acht Ziffern 0 ... 7 zur Verfügung; jede Oktalziffer entspricht drei Bits. Zur Darstellung eines Bytes werden also drei Oktalziffern benötigt, wobei von der höchstwertigen, ersten Oktalziffer nur die niederwertigen zwei Bits verwendet werden, da ein Bit übrigbleibt (3 × 3 Bit = 9 Bit). *Abb. 1.1.2.3* enthält eine Umrechnungstabelle für Hexadezimal- und Oktalzahlen.

H.	Okt.	H.	Okt.	H.	Okt.	H.	Okt.	H.	Okt.	H.	Okt.	H.	Okt.	H.	Okt.
00	000	01	001	02	002	03	003	04	004	05	005	06	006	07	007
08	010	09	011	0A	012	0B	013	0C	014	0D	015	0E	016	0F	017
10	020	11	021	12	022	13	023	14	024	15	025	16	026	17	027
18	030	19	031	1A	032	1B	033	1C	034	1D	035	1E	036	1F	037
20	040	21	041	22	042	23	043	24	044	25	045	26	046	27	047
28	050	29	051	2A	052	2B	053	2C	054	2D	055	2E	056	2F	057
30	060	31	061	32	062	33	063	34	064	35	065	36	066	37	067
38	070	39	071	3A	072	3B	073	3C	074	3D	075	3E	076	3F	077
40	100	41	101	42	102	43	103	44	104	45	105	46	106	47	107
48	110	49	111	4A	112	4B	113	4C	114	4D	115	4E	116	4F	117
50	120	51	121	52	122	53	123	54	124	55	125	56	126	57	127
58	130	59	131	5A	132	5B	133	5C	134	5D	135	5E	136	5F	137
60	140	61	141	62	142	63	143	64	144	65	145	66	146	67	147
68	150	69	151	6A	152	6B	153	6C	154	6D	155	6E	156	6F	157
70	160	71	161	72	162	73	163	74	164	75	165	76	166	77	167
78	170	79	171	7A	172	7B	173	7C	174	7D	175	7E	176	7F	177
80	200	81	201	82	202	83	203	84	204	85	205	86	206	87	207
88	210	89	211	8A	212	8B	213	8C	214	8D	215	8E	216	8F	217
90	220	91	221	92	222	93	223	94	224	95	225	96	226	97	227
98	230	99	231	9A	232	9B	233	9C	234	9D	235	9E	236	9F	237
A0	240	A1	241	A2	242	A3	243	A4	244	A5	245	A6	246	A7	247
A8	250	A9	251	AA	252	AB	253	AC	254	AD	255	AE	256	AF	257
B0	260	B1	261	B2	262	B3	263	B4	264	B5	265	B6	266	B7	267
B8	270	B9	271	BA	272	BB	273	BC	274	BD	275	BE	276	BF	277
C0	300	C1	301	C2	302	C3	303	C4	304	C5	305	C6	306	C7	307
C8	310	C9	311	CA	312	CB	313	CC	314	CD	315	CE	316	CF	317
D0	320	D1	321	D2	322	D3	323	D4	324	D5	325	D6	326	D7	327
D8	330	D9	331	DA	332	DB	333	DC	334	DD	335	DE	336	DF	337
E0	340	E1	341	E2	342	E3	343	E4	344	E5	345	E6	346	E7	347
E8	350	E9	351	EA	352	EB	353	EC	354	ED	355	EE	356	EF	357
F0	360	F1	361	F2	362	F3	363	F4	364	F5	365	F6	366	F7	367
F8	370	F9	371	FA	372	FB	373	FC	374	FD	375	FE	376	FF	377

Abb. 1.1.2.3 Umrechnungstabelle für hexadezimale und oktale Zahlen bis hex FF bzw. dezimal 255

Der BCD-Code

Eine Abart der binären Zahlendarstellung ist der sogenannte BCD-Code (Binary Coded Decimal). Ähnlich wie der Hexadezimal-Code jeder Gruppe von vier Bits eine Ziffer von 0 bis F zuordnet, ordnet man im BCD-System je vier Bits eine Dezimalziffer von 0 bis 9 zu. Damit werden natürlich nur 10 der 16 denkbaren 4-Bit-Kombinationen ausgeschöpft; die Codes, die den Hexziffern A ... F entsprechen, sind im BCD-System unzulässig.

Die Zahl 255 muß im BCD-System mit $3 \times 4 = 12$ Bits codiert werden, da sie drei Ziffern umfaßt, wovon jede in vier Bits dargestellt wird:

255 (dez.) = 0010 0101 0101 (BCD)

Im Gegensatz dazu benötigt die Dezimalzahl 255 im Hex-Code nur zwei Ziffern (FF) und acht Bits (1111 1111).

Der Vorteil der BCD-Darstellung ist, daß die Umrechnung der binären Darstellung in eine Dezimalzahl einfacher ist als bei einer rein dualen Codierung. Dem steht der Nachteil gegenüber, daß für den Mikroprozessor die vier Grundrechenarten, insbesondere aber Multiplikation und Division, im BCD-System schwieriger durchzuführen sind als dual.

Wenn man in der Mikrocomputertechnik von einer Binär-Dezimal-Umwandlung spricht, so meint man damit eigentlich meist das Umrechnen einer Dualzahl in die BCD-Darstellung, wobei dann stets zwei Dezimalziffern in einem Byte untergebracht sind.

1.1.3 Zeichendarstellung in Computern

Der ISO-7-Bit-Code

Nachdem im letzten Kapitel die Codierung von Zahlen besprochen wurde, bleibt die Frage, wie man mit einem Computer Textverarbeitung betreiben kann — wie also Buchstaben und Satzzeichen (engl. Character = Zeichen) gespeichert werden.

Das Prinzip ist dasselbe wie bei den Zahlen: Man nimmt eine Gruppe von Bits und ordnet jeder denkbaren Kombinationsmöglichkeit von Nullen und Einsen ein bestimmtes Zeichen zu. Diese Zuordnung ist natürlich im Grunde willkürlich, und in der Tat gibt es dafür mehrere Standards. Zum Glück hat sich aber nur eine Norm wirklich durchgesetzt; es ist der ISO-7-Bit-Code (ISO = International Standards Organisation). Er berücksichtigt auch die Notwendigkeit, bestimmte nationale Sonderzeichen wie ä, ö, ü, ß und § nur in Deutschland zu verwenden und in den USA durch andere, dort häufiger gebrauchte Zeichen zu ersetzen. Die US-Version des Standards heißt ASCII (American Standard Code for Information Interchange).

Da der Code sieben Bits zur Zeichendarstellung verwendet, gibt es $2\uparrow7 = 128$ mögliche Zeichen. 33 davon sind Steuerzwecken vorbehalten und werden normalerweise nicht gedruckt *(Abb. 1.1.3.1)*, die übrigen 95 sind auf einem Bildschirm oder einem Drucker sichtbar. In *Abb. 1.1.3.2* ist das erste Zeichen links oben nur scheinbar nicht sichtbar: Es handelt sich um den Leerraum (engl. Space, Hexcode 20). Ganz grob läßt sich der Code in die folgenden zwei Gruppen aufteilen:

Hex-Code	Dezimal	Zeichenart
00...1F; 7F	0...31; 127	Steuerzeichen
20...7E	32...126	Sichtbare Zeichen

H.	Abk.	Englisch	Bedeutung
00	NUL	null	Null,keine Wirkung
01	SOH	start of header	Kopfblock-Beginn
02	STX	start of text	Textbeginn
03	ETX	end of text	Textende
04	EOT	end of transmission	Ende der Sendung
05	ENQ	enquiry	Anfrage
06	ACK	acknowledge	Bestätigung
07	BEL	bell	Klingel, Piepton
08	BS	backspace	Cursor zurück
09	HT	horizontal tab	Horizontal-Tabulator
0A	LF	line feed	Cursor nach unten
0B	VT	vertical tab	Vertikal-Tabulator
0C	FF	form feed	Blattvorschub
0D	CR	carriage return	Wagenrücklauf
0E	SO	shift out	hinausschieben
0F	SI	shift in	hereinschieben
10	DLE	data link escape	Verbindung unterbrechen
11	DC1	direct control 1	Direktsteuerung 1
12	DC2	direct control 2	Direktsteuerung 2
13	DC3	direct control 3	Direktsteuerung 3
14	DC4	direct control 4	Direktsteuerung 4
15	NAK	not acknowledged	nicht bestätigt
16	SYN	synchronous idle	Synchronisierzeichen
17	ETB	end of transm.block	Ende des Sendeblocks
18	CAN	cancel	löschen
19	EM	end of medium	Speichermedium-Ende
1A	SUB	substitute	ersetzen
1B	ESC	escape	beenden/umschalten
1C	FS	form separator	Blatt-Trennung
1D	GS	group separator	Gruppentrennung
1E	RS	Record separator	Record-Trennung
1F	US	unit separator	Trennung von Einheiten

Abb. 1.1.3.1 ASCII- bzw. ISO-7-Bit-Steuerzeichen und ihre hexadezimale Codierung

Die am häufigsten gebrauchten Steuerzeichen sind:

Hex-Code	Zeichenname	Zweck
00	NUL=Null	Zeitverzögerung z.B. f. Drucker
07	BEL=Bell	Tonsignal (Piep)
08	BS=Backspace	Cursor ein Zeichen nach links
0A	LF=Line Feed	Cursor eine Zeile nach unten
0D	CR=Carriage Return	Wagenrücklauf bzw. Cursor an den Anfang der nächsten Zeile
7F	DEL=Delete	Letztes Zeichen löschen

Als Cursor bezeichnet man die aktuelle Schreibposition, d.h. jene Stelle auf dem Bildschirm, auf der das nächste Zeichen geschrieben werden wird und die gewöhnlich durch ein blinkendes Etwas (inverser Block, Pfeil, tiefliegender Strich o.ä.) erkennbar ist.

	HEX	DEZ		HEX	DEZ		HEX	DEZ		HEX	DEZ		HEX	DEZ		HEX	DEZ
	20	32	0	30	48	§	40	64	P	50	80	`	60	96	p	70	112
!	21	33	1	31	49	A	41	65	Q	51	81	a	61	97	q	71	113
"	22	34	2	32	50	B	42	66	R	52	82	b	62	98	r	72	114
#	23	35	3	33	51	C	43	67	S	53	83	c	63	99	s	73	115
$	24	36	4	34	52	D	44	68	T	54	84	d	64	100	t	74	116
%	25	37	5	35	53	E	45	69	U	55	85	e	65	101	u	75	117
&	26	38	6	36	54	F	46	70	V	56	86	f	66	102	v	76	118
'	27	39	7	37	55	G	47	71	W	57	87	g	67	103	w	77	119
(	28	40	8	38	56	H	48	72	X	58	88	h	68	104	x	78	120
)	29	41	9	39	57	I	49	73	Y	59	89	i	69	105	y	79	121
*	2A	42	:	3A	58	J	4A	74	Z	5A	90	j	6A	106	z	7A	122
+	2B	43	;	3B	59	K	4B	75	Ä	5B	91	k	6B	107	ä	7B	123
,	2C	44	<	3C	60	L	4C	76	Ö	5C	92	l	6C	108	ö	7C	124
-	2D	45	=	3D	61	M	4D	77	Ü	5D	93	m	6D	109	ü	7D	125
.	2E	46	>	3E	62	N	4E	78	^	5E	94	n	6E	110	ß	7E	126
/	2F	47	?	3F	63	O	4F	79	_	5F	95	o	6F	111			

Abb. 1.1.3.2 Hexadezimale und dezimale Codes für druckende Zeichen der deutschen Version der ISO-7-Bit-Norm (CCITT Nr. 5). Gleichzeitig können Sie diese Tabelle zum Umrechnen hexadezimaler in dezimale Zahlen und umgekehrt verwenden

Es sei darauf hingewiesen, daß insbesondere die Verwendung von Steuerzeichen, die in dieser Kurzübersicht nicht genannt sind, in Computern unterschiedlicher Hersteller sehr uneinheitlich ist, z.B. zum Löschen des Bildschirms oder zur Bewegung des Cursors nach rechts oder oben. Manchmal sind sogar einige sichtbare Sonderzeichen in jenem Codebereich zu finden, der eigentlich den unsichtbaren Steuercodes vorbehalten ist. In einem Wort: Es gibt zahlreiche Computer, die nur bedingt dem ISO-Zeichenstandard bzw. der ASCII-Norm entsprechen. Dies gilt zum Teil sogar für die sichtbaren (die „druckenden") Zeichen. So sind z.B. bei Commodore-Computern die Kleinbuchstaben an der Stelle der Codetabelle, wo eigentlich die Großbuchstaben hingehören, und die Großbuchstaben werden mit einem achten Codebit angewählt, das bei der ISO-Norm gar nicht vorgesehen ist.

In der US-ASCII-Norm stehen anstelle der deutschen Sonderzeichen folgende Zeichen:

Hex-Code	Dez.-Code	Deutsches Zeichen	US-ASCII-Zeichen
40	64	§	@
5B	91	Ä	[
5C	92	Ö	\
5D	93	Ü	]
7B	123	ä	{
7C	124	ö	|
7D	125	ü	}
7E	126	ß	~

In anderen europäischen Ländern, z.B. England, Frankreich und Skandinavien, werden teilweise wiederum andere Zeichen für diese Codes verwendet.

Das achte Bit

Da Mikrocomputer normalerweise mit 8-Bit-Worten (Bytes) arbeiten, bleibt das achte Bit sozusagen übrig. Es dient oftmals aber zur Umschaltung auf einen alternativen Zeichensatz (z.B. deutsch/ASCII) oder auf einen Satz von Grafikzeichen. Bei der Übertragung per Telefon oder bei der Abspeicherung z.B. auf Kassetten kann es ferner als Paritäts-Prüfbit zur Fehlererkennung dienen (engl. parity). Dazu summiert man die Eins-Bits des Zeichens: Wenn die Summe gerade ist, setzt man das Parity-Bit auf Null, wenn sie ungerade ist, auf Eins (ungerade Parität = engl. odd parity). Manchmal verwendet man auch das umgekehrte Verfahren und setzt das Parity-Bit auf Eins, wenn die Summe der übrigen Bits gerade ist (gerade Parität = engl. even parity).

Der 5-Bit-Telex-Code

Der internationale Fernschreib-Verkehr wird zum großen Teil noch im ISO-5-Bit-Code abgewickelt, landläufig auch Baudot-Code genannt. Da man mit fünf Bits nur 32 unterschiedliche Zeichen darstellen kann, die nicht für alle Buchstaben, Ziffern und Satzzeichen ausreichen würden, hat man im 5-Bit-Code zwei „Zeichenebenen" definiert — eine Buchstaben- und eine Zeichenebene. Besondere Steuerzeichen dienen zur Umschaltung zwischen beiden Zeichengruppen (BU für die Buchstabenebene und ZI für die Ziffern und Satzzeichen).

Die 32 Zeichen der Buchstaben-Ebene reichen leider nicht für eine gemischte Groß- und Kleinschreibung aus. Baudot-Fernschreiber drucken daher entweder nur große oder nur kleine Buchstaben. Auch der Vorrat an Satzzeichen ist recht begrenzt *(Abb. 1.1.3.3)*.

```
Buchstaben-Ebene:                    Zeichen in beiden Ebenen:

A 03   B 19   C 0E   D 09            04 Leerraum (Space)
E 01   F 0D   G 1A   H 14            02 Zeilenvorschub (LF)
I 06   J 0B   K 0F   L 12            08 Wagenrücklauf (CR)
M 1C   N 0C   O 18   P 16            00 Null (keine Wirkung)
Q 17   R 0A   S 06   T 10            1F Buchstaben-Ebene
U 07   V 1E   W 13   X 1D            1B Ziffern-Ebene
Y 15   Z 11

Ziffern/Zeichen-Ebene:

! 14   # 09   & 1A   ' 05
( 0F   ) 12   * 19   + 11
, 0C   - 03   . 1C   / 1D
0 16   1 17   2 13   3 01
4 0A   5 10   6 15   7 07
8 06   9 18   : 0E   ; 0B
= 1E   ? 19
```

Abb. 1.1.3.3 Beim Baudot-Code (CCITT Nr. 2) sind zwei Zeichenebenen vorhanden, zwischen denen mit besonderen Steuerzeichen umgeschaltet werden kann. Dadurch kommt man mit fünf Bits als Code aus (hexadezimal 00...1F)

40	4B .	4D (	4E +	50 &	5B $
5C *	5D)	5E ;	6C %	6D -	6F ?
7A :	7B #	7C §	7D ,	7E =	7F "
81 a	82 b	83 c	84 d	85 e	86 f
87 g	88 h	89 i	91 j	92 k	93 l
94 m	95 n	96 o	97 p	98 q	99 r
A2 s	A3 t	A4 u	A5 v	A6 w	A7 x
A8 y	A9 z	C1 A	C2 B	C3 C	C4 D
C5 E	C6 F	C7 G	C8 H	C9 I	D1 J
D2 K	D3 L	D4 M	D5 N	D6 O	D7 P
D8 Q	D9 R	E2 S	E3 T	E4 U	E5 V
E6 W	E7 X	E8 Y	E9 Z	F0 0	F1 1
F2 2	F3 3	F4 4	F5 5	F6 6	F7 7
F8 8	F9 9				

Abb. 1.1.3.4
Druckende Zeichen im EBCDIC-Code, der vor allem von IBM-Großrechnern verwendet wird, aber nicht offiziell genormt ist

Der EBCDIC-Zeichensatz

Vorwiegend im Großrechner-Bereich ist auch der ursprünglich von IBM entwikkelte EBCDIC-Standard verbreitet (Extended Binary Coded Decimal Information Code = erweiterter BCD-Informationscode). Er arbeitet mit acht Bits für jedes Zeichen. Da aber keine $2^8 = 256$ Zeichen nötig sind, gibt es eine ganze Reihe von nicht zulässigen Bitkombinationen. Die erlaubten Hex-Codes sind in *Abb. 1.1.3.4* zusammengestellt.

Der EBCDIC-Standard spielt allerdings in der Mikrocomputer-Technik im Vergleich mit der ISO-7-Bit-Norm so gut wie keine Rolle. Auch im Großrechner-Bereich wird er immer weniger verwendet, so daß auf ihn nicht näher eingegangen werden muß.

1.2 Mikroprozessoren

1.2.1 Überblick

Da das Rechnen mit Kopf, Bleistift und Papier nicht nur anstrengend, sondern auch fehlerträchtig ist, kam man schon früh auf die Idee, Hilfsmittel dafür zu verwenden. Den Abakus, den man heute noch in manchen asiatischen Ländern oft sieht, gab es schon 1100 v. Chr. Eine „richtige" mechanische Rechenmaschine für Addition und Subtraktion baute 1652 der Franzose Blaise Pascal; sie arbeitete mit Einstellrädern für Dezimalziffern.

Die mathematische Voraussetzung zum Bau binär arbeitender Rechner schuf Gottfried Wilhelm Leibniz 1674 mit einem mechanischen „Computer", der nur die Ziffern 0 und 1 verwendete und immerhin die vier Grundrechenarten beherrschte. Auf dem gleichen Prinzip basierte der erste programmierbare Rechner, den Kon-

rad Zuse 1935 aus elektromechanischen Relais baute. 1946 entstand in den USA ein Computer aus 1800 Elektronenröhren, der „Eniac", bis drei Jahre später die beiden Forscher Bardeen und Brattain in den Bell-Laboratorien den Transistor erfanden.

Die technologische Voraussetzung zur Herstellung solch komplexer Gebilde wie eines kompletten Prozessors auf einem einzigen Silizium-Chip geht aber auch auf Jack Kilby zurück, der 1958 bei Texas Instruments das IC (integrated circuit, Integrierte Schaltung) erfand. Das erste IC besaß nur vier Transistoren auf dem Chip; moderne Mikroprozessoren enthalten einige Zehntausend, teilweise sogar über 100000 auf einem Siliziumplättchen von weniger als einem Quadratzentimeter Fläche.

Bei Mikroprozessoren spricht man von Generationen, die im Abstand weniger Jahre wechselten. Etwa gleichzeitig haben um 1970 die Firmen Intel (unter geistiger Vaterschaft eines gewissen Ted Hoff) und Texas Instruments erstmals zwei 4-Bit-Mikroprozessoren vorgestellt, d.h. der Datenbus war vier Bit „breit". Größere Verbreitung erlangte allerdings nur der TI-Typ TMS-1000, während sich Intel bald auf die Entwicklung eines 8-Bit-Prozessors konzentrierte, des 8008. Er war 1972 fertig. Dessen Nachfolgertyp, der 1974 entwickelte 8080, war lange Zeit der Industrie-Standard der 8-Bit-Welt und wurde von zahlreichen Zweitherstellern nachgebaut.

Nahezu zeitgleich mit dem 8080 entstanden der 2650 von Valvo, der SC/MP von National Semiconductor, der F8 von Fairchild, der 3870 von Mostek und, mit fast ähnlichem Erfolg wie der 8080, der 6800 von Motorola. Das war sozusagen die zweite Generation.

Die dritte Generation entstand 1976 mit zwei Abkömmlingen der erfolgreicheren Prozessoren 8080 und 6800. Ehemalige Intel-Mitarbeiter gründeten die Firma Zilog und brachten den Z80 heraus. Und frühere Motorola-Mitarbeiter gründeten MOS Technology (diese Firma wurde nach einigen Jahren von Commodore aufgekauft) und entwickelten den 6502. Schon 1982 waren Z80 und 6502 die am weitesten verbreiteten 8-Bit-Mikros, und man sieht ihnen ihre Herkunft teilweise noch recht deutlich an.

Lange vorher gab es sogar schon 16-Bit-Mikroprozessoren: National Semiconductor brachte 1975 den „Pace" und Texas Instruments 1976 den TMS-9900 heraus. Beide waren offensichtlich ihrer Zeit voraus, denn keiner von ihnen fand große Verbreitung. Größerer Erfolg war erst Intels 8086 beschieden, der 1979 herauskam, und ab 1982 dem 68000 von Motorola.

Bei der Besprechung der internen Struktur der unterschiedlichen Prozessortypen wird die zeitliche Entwicklung deutlich werden, insbesondere beim 68000: Der Trend geht zu einer einfacheren, überschaubareren Programmierbarkeit, zu einer strukturellen Unterstützung bei der Implementation höherer Programmiersprachen und zu einer effizienten Verwaltung auch sehr großer Arbeitsspeicher. So gesehen, könnte man Prozessoren wie 68000, Z8000 und NS16032 zur vierten

Generation rechnen, während dem 8086 noch zahlreiche Beschränkungen der vorherigen 8-Bit-Generation eigen sind.

An dieser Stelle noch ein Wort über die Definition, was ein 8-Bit- und was ein 32-Bit-Prozessor ist. Hierzu gibt es unterschiedliche Meinungen: Manche sagen, dies hänge von der Breite der internen Datenregister ab. Dann müßte man die 8-Bit-CPUs 6800 oder Z80 als 16-Bit-Prozessoren bezeichnen. Andere sind der Meinung, die Rechenlogik im Prozessor sei dafür ausschlaggebend: Dann wäre der Z80 ein 4-Bit-Prozessor, da er eine 8-Bit-Addition intern in zwei 4-Bit-Hälften ausführt. Das wohl einzig vernünftige Kriterium ist daher die Zahl der externen Datenbus-Leitungen. Prozessoren wie 68008 und 8088, die die intern verwendeten 32- bzw. 16-Bit-Worte in Teile zu je 8 Bit aufspalten und nacheinander auf den Datenbus geben, muß man daher konsequenterweise als 8-Bit-Prozessoren bezeichnen, obwohl sie die gleichen Befehle verstehen wie ihre „echten" 16-Bit-Kollegen 68000 und 8086, wenn auch bei geringerer Verarbeitungs-Geschwindigkeit.

Auch diese Aussage beinhaltet allerdings nicht immer die ganze Wahrheit. Es gibt eine Reihe von Prozessortypen, die Adressen und Daten abwechselnd auf gemeinsame CPU-Anschlüsse leiten, d.h. multiplexen (8085, 8086, Z8000 u.a.). Diese Maßnahme ermöglicht Chipgehäuse mit weniger Pins und führt deshalb zu preisgünstigeren CPUs. Allerdings sind dafür extern „Demultiplexer"-ICs nötig, die die Daten wieder von den Adressen trennen. Ferner bedingt das Multiplexen einen zeitkritischeren und langsameren Speicherzugriff als bei CPUs, bei denen Adressen und Daten auf getrennte Pins einzeln herausgeführt sind (z.B. 68000, 68020).

Die Nomenklatur dieses Buches bezeichnet, übrigens in Übereinstimmung mit der Literatur der Prozessor-Hersteller, CPUs wie 8085, 6800, Z80, 6502, 8088 und 68008 als 8-Bit-Prozessoren und 8086, Z8000, 68000 und NS16032 als 16-Bit-CPUs. Computerhersteller, die diese Prozessoren in ihren Produkten einsetzen, sprechen werbewirksam statt 8 Bit oft von 16 Bit und statt 16 Bit von 32 Bit, was hier aber nicht guten Gewissens vertreten werden kann.

In der *Abb. 1.2.1.1* sind jene handelsüblichen Mikroprozessoren aufgeführt, die in der Standard-NMOS-Technologie hergestellt werden. Sie verwendet gewöhnlich eine einzige Versorgungsspannung von +5 V. Der Stromverbrauch dieser Prozessoren liegt bei etwa 50...200 mA. Für batteriebetriebene Geräte eignen sich daher besser CPUs, die in der stromsparenden CMOS-Technologie gefertigt werden *(Abb. 1.2.1.2)*; auch sie arbeiten meist mit +5 V, besitzen aber neben ihrem geringeren Leistungsbedarf (etwa nur ein Zehntel bis ein Drittel von NMOS-CPUs, das hängt von der Taktfrequenz ab) auch den Vorteil einer geringeren Empfindlichkeit gegenüber Störimpulsen aus der Umgebung. Allerdings ist für die gleiche Anzahl logischer Funktionen eine größere Chipfläche nötig, was CMOS-Prozessoren gegenüber ihren NMOS-Brüdern teurer macht. Früher war auch die PMOS-Technik (Hochvolt-Technologie) verbreitet (z.B. TMS-1000), spielt heute aber kaum noch eine Rolle.

Hersteller	Typ	Bits/ Wort	Adressen-raum/KByte
Commodore	650X	8	64
Commodore	651X	8	64
Data General	MN602	16	64
DEC	DCJ11	16	32
Fairchild	F8	8	64
Fujitsu	6800	8	64
Fujitsu	6802	8	64
Fujitsu	6809	8	64
Fujitsu	8088	8	1024
Fujitsu	8088	16	1024
Intel	8031	8	128
Intel	8035	8	64
Intel	8039	8	64
Intel	8080A	8	64
Intel	8085	8	64
Intel	8086	16	1024
Intel	8087	80/16	1024
Intel	8088	8	1024
Intel	80186	16	1024
Intel	80286	16	16384
Intel	432	32	16384
LSI Systems	7270	12	4
Matra-Harris	8035H	8	4
Matra-Harris	8031	8	64
Matra-Harris	8086	16	1024
Matra-Harris	8088	8	1024
Matsushita	1610	16	64
Matsushita	1613	16	256
Mostek	3870	8	64
Mostek	3873	8	64
Motorola	6800	8	64
Motorola	6802	8	64
Motorola	6803	8	64
Motorola	6809	8	64
Motorola	68000	16	16384
Motorola	68008	8	1024
Motorola	68010	16	16384
Motorola	68020	32	16384
National Sem.	8060	8	64
National Sem.	8040	8	64
National Sem.	8070	8	64
National Sem.	16008	8	16384
National Sem.	16016	16	16384
National Sem.	16032	16	65536
National Sem.	32032	32	16384
NEC	7800	8	64
Rockwell	650X	8	64
Rockwell	651X	8	64
Rockwell	68000	16	16384
Siemens	8031	8	128
Siemens	8035	8	64
Siemens	8080	8	64
Siemens	8085	8	64
Siemens	8086	16	1024
Siemens	80286	16	16384
Signetics	2650	8	32
Synertek	650X	8	64
Texas Instrum.	7000	8	64
Texas Instrum.	9900	16	64
Texas Instrum.	9980	8	16
Texas Instrum.	9995	8	16
Texas Instrum.	99000	16	64
Toshiba	4300	4	2
Toshiba	4600	4	4
Toshiba	8035	8	64
Toshiba	8039	8	64
Toshiba	8080	8	64
Toshiba	8085	8	64
Western Digit.	WD16	16	64
Zilog	Z80	8	64
Zilog	Z800	8	128
Zilog	Z8001	16	49152
Zilog	Z8002	16	354
Zilog	Z8003	16	49152
Zilog	Z8004	16	354

Abb. 1.2.1.1
NMOS-Mikroprozessoren

Hersteller	Typ	Bits/ Wort	Adressen-raum/KByte
AMI	S99C91	16	64
AMI	AMU/PR	16	64
Commodore	65C0X	8	64
Commodore	65C1X	8	64
DEC	DCT11	16	32
Fujitsu	MB8850	8	2
Fujitsu	MB8859	8	2
GTE Microcirc.	65SC0X	8	64
GTE Microcirc.	65SC1X	8	64
GTE Microcirc.	65SC1XX	8	64
GTE Microcirc.	65SC816	16	16384
Harris	6120	12	64
Harris	80C86	16	1024
Hitachi	6301	8	64
Intersil	IM6100	12	4
Matra-Harris	HM80C35	8	4
Mitel	68SC02	8	64
Motorola	14500	1	0
Motorola	146805	8	8
National Semic.	NSC800	8	64
NEC	7500G	4	8
NEC	75C08	4	4
NEC	78C05	8	64
NEC	7810	8	64
NEC	5840H	4	4
NEC	80C35	8	64
NEC	80C85	8	64
Oki	80C35	8	64
Oki	80C39	8	64
Plessey	68SC02	8	64
RCA	CDP1802	8	64
RCA	CDP1805	8	64
RCA	CDP1806	8	64
RCA	CDP6805	8	8
Rockwell	65C0X	8	64
Rockwell	65C1X	8	64
Sharp	LH5080	8	64
Texas Instrum.	1000JLC	4	1
Texas Instrum.	1100JLC	4	2
Toshiba	4600AC	4	4
Toshiba	80C35	8	64
Toshiba	80C39	8	64

Abb. 1.2.1.2 CMOS-Mikroprozessoren

Wie schon erwähnt, ist es auch möglich, RAM, ROM und Ein/Ausgabe-Funktionen gleich mit auf dem Prozessor-Chip zu integrieren. Man spricht dann von Einchip-Computern. *Abb. 1.2.1.3* nennt solche Chips in NMOS- und *Abb. 1.2.1.4* in CMOS-Technik.

Hersteller	Typ	RAM/ Bits	ROM/ Bits, E=EPROM
AMS	528211	256x16	512x18
Fairchild	3870	64x8	2048x8
Fujitsu	8841	128x4	2048x8
Fujitsu	88401	192x4	4096x8
Fujitsu	6801	128x8	2048x8
Fujitsu	6801W	192x8	4096x8
Fujitsu	8049	128x8	2048x8
General Instr.	PIC650	32x8	512x12
Hitachi	6801S	128x8	2048x8
Hitachi	6801V	128x8	4096x8
Hitachi	6805S	64x8	1100x8
Hitachi	6805U	96x8	2056x8
Hitachi	6805W	96x8	3848x8
Intel	8020	64x8	1024x8
Intel	8021	64x8	1024x8
Intel	8022	64x8	2048x8
Intel	8041	64x8	1024x8
Intel	8741	64x8	E1024x8
Intel	8048	64x8	1024x8
Intel	8748	64x8	E1024x8
Intel	8049	128x8	2048x8
Intel	8749	128x8	E2048x8
Intel	8051	128x8	4096x8
Intel	8751	128x8	E4096x8
Intel	8096	116x16	4096x16
Intel	2920	40x25	192x24
Matra-Harris	8048H	64x8	1024x8
Matra-Harris	8051H	128x8	4096x8
Matsushita	1400	64x4	1024x8
Matsushita	1405	128x4	2048x8
Mitsubishi	8050H	256x8	4096x8
Mostek	3870	128x8	4032x8
Mostek	3875	128x8	4096x8
Mostek	68200	128x16	2048x16
Motorola	6801	128x8	2048x8
Motorola	68701	128x8	E2048x8
Motorola	6801U	192x8	4096x8
Motorola	68701U	192x8	E4096x8
Motorola	6805P	64x8	1100x8
Motorola	6805R	64x8	2048x8
Motorola	6805T	64x8	2500x8
Motorola	6805U	648x8	2048x8
Motorola	68705P	112x8	E1804x8
Motorola	68705R	112x8	E3776x8
Motorola	68705U	112x8	E3776x8
National Sem.	COP410	32x4	512x8
National Sem.	COP420	64x4	1024x8
NEC	7801	128x8	4096x8
NEC	7802	128x8	6144x8
NEC	7811	128x8	4096x8
Rockwell	6500/1	64x8	2048x8
Siemens	8021	64x8	1024x8
Siemens	8048	64x8	1024x8
Siemens	8051	128x8	4096x8
Signetics	8021	64x8	1024x8
Signetics	8041	64x8	1024x8
Signetics	8048	64x8	1024x8
Signetics	8049	128x8	2048x8
Synertek	Z8	64x8	1024x8
Texas Instrum.	7020	128x8	2048x8
Texas Instrum.	7040	128x8	4096x8
Texas Instrum.	9940	128x8	64x16
Texas Instrum.	TMS320	144x16	1536x16
Toshiba	4310	48x4	1024x8
Toshiba	4320	128x4	2048x8
Toshiba	8022	64x8	1024x8
Toshiba	8048	64x8	1024x8
Toshiba	8049	128x8	2048x8
Toshiba	8051	128x8	4096x8
Zilog	Z8601	144x8	2048x8
Zilog	Z8611	144x8	4096x8

Abb. 1.2.1.3
NMOS-Einchip-Mikrocomputer

Hersteller	Typ	RAM Bits	ROM Bits
Fairchild	38C70	64x8	2048x8
Fujitsu	8851	128x4	2048x8
Fujitsu	8853	64x4	1024x8
Fujitsu	88504	192x4	4096x8
Fujitsu	80C49	128x8	2048x8
General Instr.	16C55	32x8	512x12
GTE Microcirc.	65SC150	64x8	2048x8
Hitachi	HD6301	128x8	4096x8
Hitachi	HD6305	96x8	3772x8
Intel	80C48	64x8	1024x8
Intel	80C49	128x8	2048x8
Intel	80C51	128x8	4096x8
ITT	SAA6000	96x4	2268x8
Matra-Harris	80C48	64x8	1024x8
Matra-Harris	80C49	128x8	2048x8
Mitsubishi	80C49	128x8	2048x8
Motorola	141000	64x4	1024x8
Motorola	146805	112x8	2048x8
Motorola	1468705	112x8	2048x8
National Semic.	COP320C	64x4	1024x8
National Semic.	COP321C	64x4	1024x8
National Semic.	COP420C	64x4	1024x8
National Semic.	COP421C	64x4	1024x8
National Semic.	80C48	64x8	1024x8
National Semic.	80CX49	128x8	2048x8
National Semic.	87PC48	64x8	1024x8
NEC	µPD650	96x4	2000x8
NEC	µPD7503	224x4	4096x8
NEC	80C48	64x8	1024x8
NEC	80C49	128x8	2048x8
Oki	6404	256x4	4096x8
Oki	80C48	64x8	1024x8
Oki	80C49	128x8	2048x8
Matsushita	MN1450	64x4	1024x8
RCA	CDP1804	64x8	2048x8
RCA	6805F2	64x8	1089x8
RCA	6805G2	112x8	2106x8
Siemens	80C482	64x8	2048x8
Synertek	Z86C01	128x8	2040x8
Texas Instrum.	1000C	64x4	1024x8
Texas Instrum.	1004C	256x4	1024x8
Texas Instrum.	1300C	128x4	2048x8
Toshiba	47C40	256x4	4096x8
Toshiba	80C48	64x8	1024x8
Toshiba	80C49	128x8	2048x8

Abb. 1.2.1.4 CMOS-Einchip-Mikrocomputer

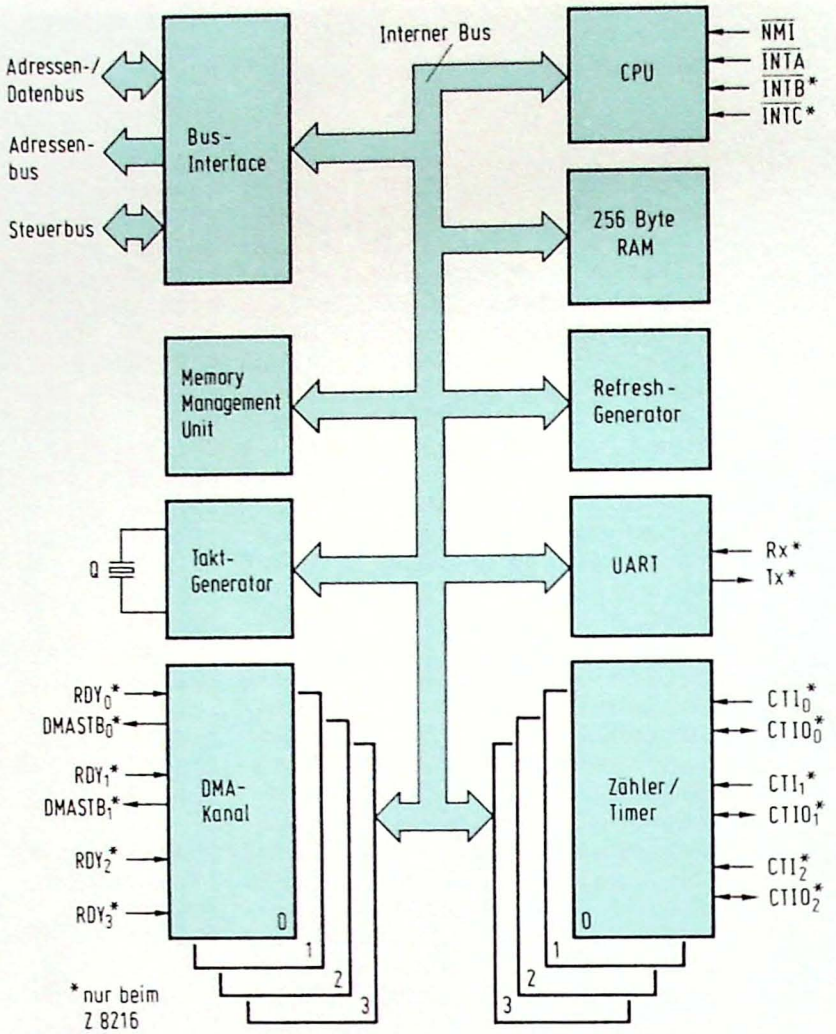


Abb. 1.2.1.5 Innere Struktur des 8-Bit-Prozessors Z800 von Zilog

Bestimmte Gruppen von Prozessortypen und Einchip-Computern verstehen die gleichen Befehle, sie sind „softwarekompatibel". Solche Gruppen sind z. B.:

3870, F8;
6800, 6801, 6802, 6805, 6301, 68705;
8080, 8085;
8031, 8035, 80C31, 80C35, 8048, 80C48, 87C48;
8088, 8086;
Z80, NSC800, LH5080;
65XX, 65CXX, 65SCXX;
9900, 9980, 9995

Dies gilt teilweise allerdings nur mit kleinen Einschränkungen. So besitzt der von NEC gefertigte 8080 ein etwas anderes Statusregister gegenüber dem Origi-

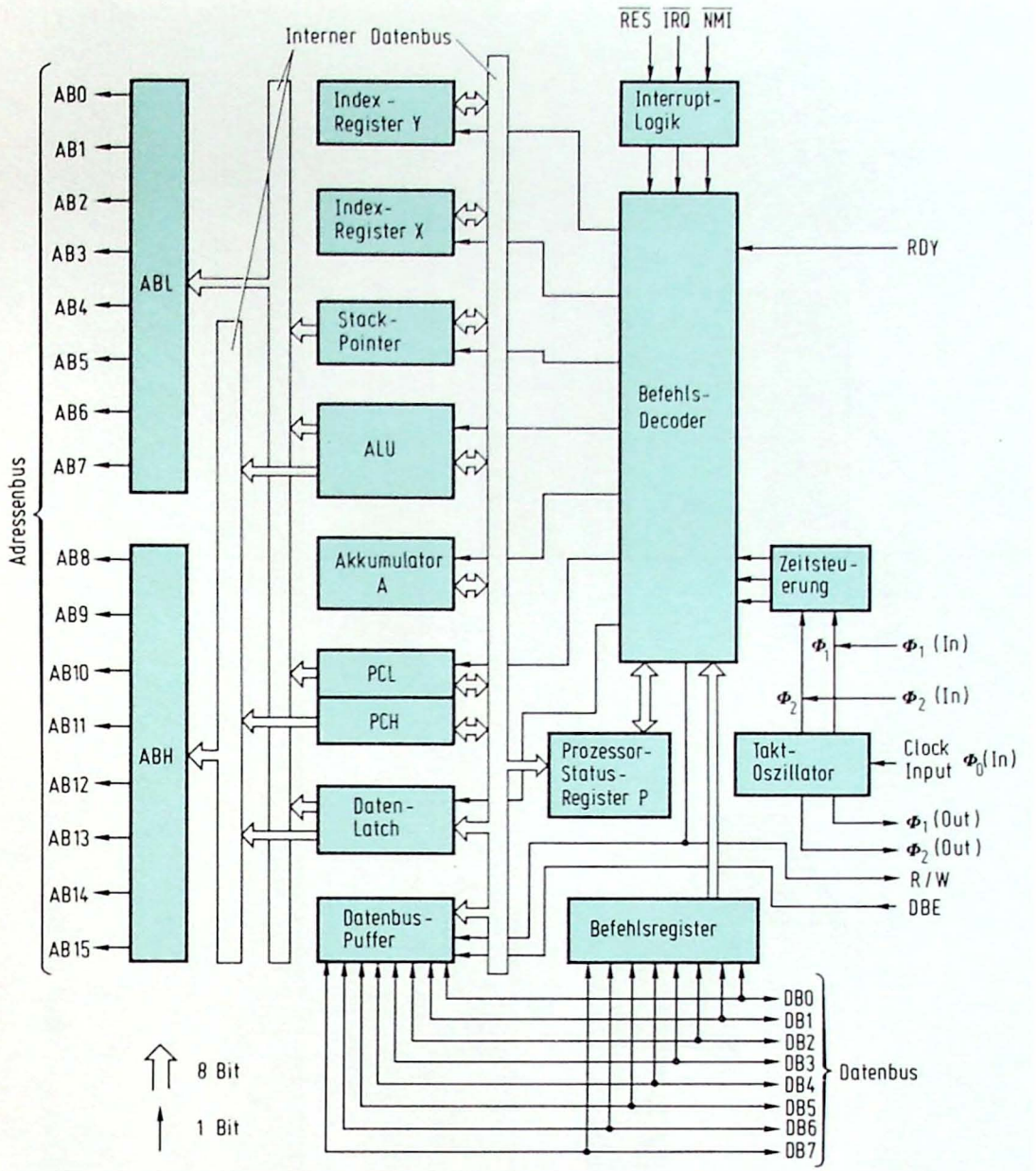


Abb. 1.2.1.6 Innenaufbau der Prozessorfamilie 65XX (ALU = Arithmetic/Logic Unit = Rechenwerk)

nal-Intel-Typ, und der 65SC02 von GTE besitzt einige kleine Abweichungen im Befehlssatz vom Rockwell-65C02. Letzterer wiederum hat einige Befehle mehr als die NMOS-Version 6502. (Ein X in obigen Übersichten bedeutet, daß es verschiedene Prozessor-Versionen innerhalb der „Familie" gibt, die sich z.B. in der Gehäuseform unterscheiden, was durch eine Ziffer an der Stelle des X gekennzeichnet wird.)

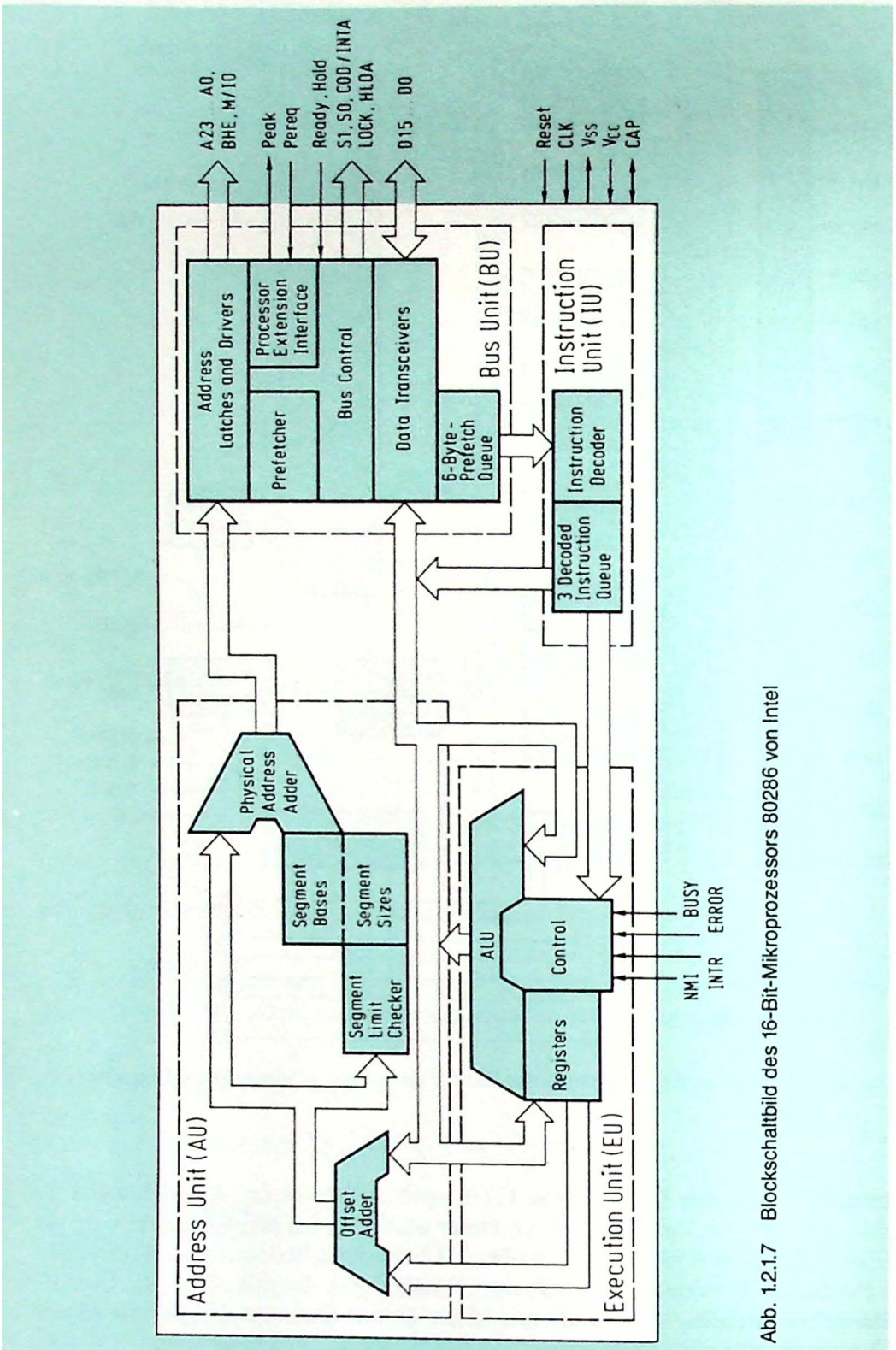


Abb. 1.2.1.7 Blockschaltbild des 16-Bit-Mikroprozessors 80286 von Intel

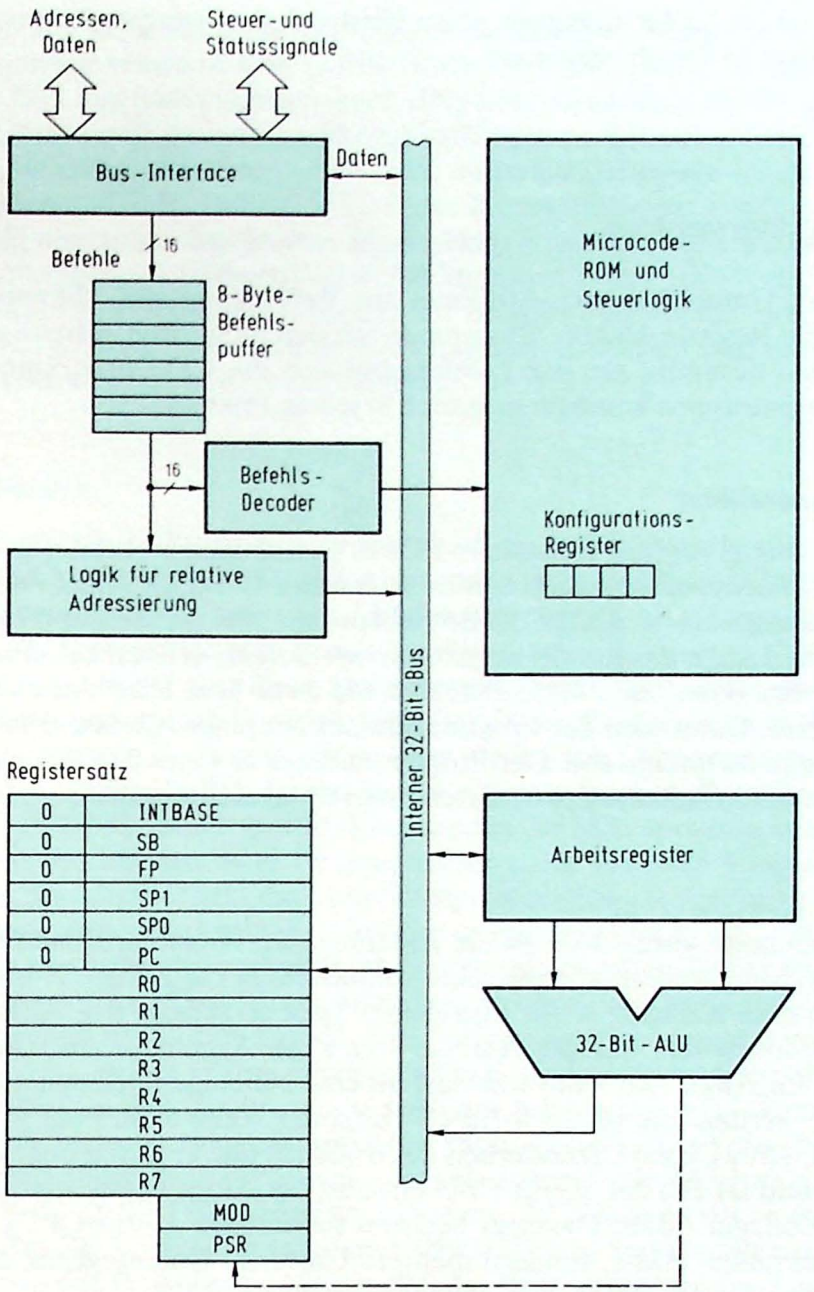


Abb. 1.2.1.8 Aufbau des Prozessors 32032 (Serie 32) von National Semiconductor

Darüber hinaus gibt es CPUs, deren Befehlssatz eine Erweiterung desjenigen von früher auf den Markt gekommenen Prozessoren ist. Der Z80 beherrscht alle 8080-Befehle, hat aber mehr Register als jener und deshalb auch einige nützliche zusätzliche Befehle. Die 16-Bit-CPU 65SC816 kann alle Befehle des 6502 ausführen, da sie sich in einen 8-Bit-Modus schalten läßt, besitzt jedoch ebenfalls mehr Register und mehr Befehle gegenüber dem Vorläufertyp.

Abb. 1.2.1.5 bis 1.2.1.8 vermitteln einen Eindruck des internen Aufbaus von zwei 8-Bit- und zwei 16-Bit-Prozessoren.

1.2.2 Registerstruktur

Wesentliches Unterscheidungs-Merkmal zur Beurteilung von Mikroprozessoren ist die interne Registerstruktur. Zusammen mit den möglichen Adressierungsarten und Befehlen bestimmt sie, wie komfortabel sich die CPU programmieren läßt. Gewisse Registertypen kommen praktisch in jedem Prozessor vor.

Der Programmzähler

Dieses Register (Program Counter = PC) ist gewöhnlich so breit wie der Adressenbus des Mikroprozessors, bei 8-Bit-CPUs meist 16 Bit. Bei einem Reset wird es mit der Anfangsadresse des ROM-Betriebsprogramms geladen und dann schrittweise um die Länge des jeweils abgearbeiteten Befehls erhöht: bei einem 1-Byte-Befehl um ein Byte, bei 2-Byte-Befehlen um zwei usw. Eine Ausnahme bilden Sprungbefehle: Dann wird der Programmzähler mit jener Adresse geladen, an der das Programm fortfahren soll. Der Programmzähler ist eines der wichtigsten CPU-Register, da es die Adresse des nächsten Programmbefehls enthält.

Der Akku

Der Akkumulator, kurz Akku, ist ein Datenregister, in dem arithmetische Operationen ausgeführt werden können. Um beispielsweise die Zahlen X und Y zu addieren, lädt man zunächst X aus irgendeiner Speicherzelle in den Akku und führt einen Additionsbefehl mit der Adresse von Y als Argument aus. Das Ergebnis steht dann im Akku und kann von dort an eine beliebige Speicherzelle im RAM transferiert werden. Bei reinen 8-Bit-CPUs ist der Akku 8 Bit breit (8080, 8085, Z80, 6800, 6502). Eine Besonderheit des 6800 ist, daß er gleich über zwei Akkus verfügt (A und B). Bei der 16-Bit-CPU 8086 ist der Akku 16 Bit breit, auch bei ihrer 8-Bit-Kollegin 8088. Dagegen besitzen modernere 16-Bit-CPUs überhaupt keinen bestimmten Akku, sondern mehrere Universal-Datenregister, die sich alle wie Akkus verwenden lassen (z.B. 68000, Z8000).

Das Statusregister

Jedes einzelne Bit des Statusregisters dient als „Flag", als Merker, ob eine bestimmte Situation eingetreten ist — beispielsweise, ob bei einer Addition ein Übertrag auftrat, weil das Ergebnis nicht mehr in den Akku paßt, oder ob das Ergebnis einer arithmetischen Operation Null, positiv oder negativ ist. Einfachere CPUs haben für diesen Zweck nicht einmal ein richtiges Register von ein oder zwei Byte

Länge, sondern tatsächlich nur ein paar 1-Bit-Speicherzellen, die irgendwo auf dem Siliziumchip verstreut sind (TMS-1000, IM 6100, 1802). In anderen Fällen sind einige Bits des Statusregisters auch direkt an Anschlüsse des Prozessors herausgeführt, um damit externe Logikpegel abfragen zu können oder irgend etwas ein- und ausschalten zu können, ohne dafür einen Ein-/Ausgabe-Baustein verwenden zu müssen (SC/MP, 1802, 2650). Jeder Prozessor besitzt bestimmte Befehle, die abhängig vom Inhalt bestimmter Status-Flags etwas tun, wie etwa bedingte Programm-Verzweigungen (Branch) oder Addition mit Carry (Übertrag). Das funktioniert ebenso wie bei einer Addition mit Bleistift und Papier: Wenn bei einer Stelle des Additions-Ergebnisses ein Übertrag auftrat, so muß man diesen zum Ergebnis der nächsthöheren Stelle dazuzählen.

Der Stackpointer

Wenn ein bestimmter Programmteil innerhalb eines Gesamtprogramms mehrere Male unverändert benötigt wird, so führt man ihn zweckmäßig als Unterprogramm aus und braucht ihn dann nur einmal in den Speicher zu schreiben. Er muß dann mit einem Rücksprung-Befehl (Return) enden. Immer, wenn man das Unterprogramm (die Subroutine) vom Hauptprogramm aufrufen will, tut man das mit einem speziellen Unterprogramm-Sprungbefehl. Damit der Prozessor nach Ausführung des Unterprogramms noch weiß, an welcher Stelle er das Hauptprogramm verlassen hat, muß er sich natürlich die Rücksprungadresse merken. Bis auf wenige Ausnahmen (TMS-1000, IM 6100, 1802) verwenden die Mikroprozessoren zu diesem Zweck einen bestimmten, vom Programmierer unter gewissen Einschränkungen festlegbaren Speicherbereich, der vom Stackpointer (Stapelzeiger) adressiert wird. Dadurch sind auch verschachtelte Unterprogramme möglich, d.h. ein Unterprogramm kann wiederum ein anderes Unterprogramm aufrufen. Dabei wandert beim Aufruf der Stackpointer im Adressenbereich von oben nach unten und beim Rücksprung von unten nach oben zurück. Der Stackpointer kann unterschiedlich lang sein: Beim 2650 umfaßt er drei Bits und adressiert einen 8 × 15-Bit-Speicher auf dem CPU-Chip, so daß der Prozessor sich acht Adressen merken kann bzw. acht verschachtelte Unterprogramme ausführen kann. Beim 6502 ist der Stackpointer 8 Bits lang und adressiert das byteweise organisierte externe RAM im Adressenbereich hex 0100...01FF. Diese 256 Bytes können 128 Adressen von je 16 Bit Länge speichern. Bei Z80, 6800 und anderen CPUs ist der Stackpointer dagegen genauso lang wie der Programmzähler, so daß der Programmierer den Stack in einen beliebigen (RAM-)Adressenbereich legen kann.

Sonstige Register

Die oben genannten Register gibt es praktisch bei jedem Prozessor. Alle anderen sind dagegen spezifisch für einen bestimmten CPU-Typ. Dazu gehören z.B. Indexregister, die es gestatten, auf bequeme Weise Datentabellen im Speicher nach einem bestimmten Wert zu durchsuchen oder eine Codewandlung durchzuführen.

Es wurde bisher stillschweigend davon ausgegangen, daß sich die Register als Speicherzellen auf dem CPU-Chip selbst befinden. Das ist keineswegs selbstverständlich, und es gibt tatsächlich eine Ausnahme von dieser Regel. Der 16-Bit-Prozessor TMS-9900 besitzt nur einen Workspace-Pointer, das Statusregister und den Programmzähler auf dem Chip, während die anderen 15 Register über den Workspace-Pointer (Arbeitsbereich-Zeiger) adressiert werden, also im RAM liegen. Dabei speichert R11 die Unterprogramm-Rücksprungadressen. Das hat den Vorteil, daß es bei Unterprogrammen oder Interrupts (Unterprogrammaufrufe, die per Hardware-Impuls statt Software-Befehl eingeleitet werden) recht einfach ist, Registerinhalte, die jetzt für einen anderen Zweck verwendet werden sollen, zu „retten": Man braucht ja nur den Workspace-Pointer auf einen anderen Adressenbereich zeigen zu lassen (Texas Instruments bezeichnet das als „Context Switch"). Der Nachteil der ins RAM ausgelagerten Register ist, daß hier auch bei Registeroperationen jedesmal ein externer Speicherzugriff nötig ist, der natürlich langsamer abläuft als ein Zugriff auf Register auf dem CPU-Chip selbst.

In Wahrheit befinden sich auf dem Silizium-Chip eines Prozessors noch wesentlich mehr Register als die oben genannten. Sie dienen jedoch nur dem internen Datentransfer, und der Programmierer braucht sich um sie nicht zu kümmern. Die Prozessorhersteller haben deshalb den Begriff des „Programmiermodells" eingeführt, das lediglich die für den Programmierer relevanten Register darstellt. Dem schließt sich dieses Buch an, weil es ja nicht für die Chip-Entwickler, sondern für die Anwender von Prozessoren gedruckt wurde.

Die Registerlänge

Die Länge der einzelnen Register hat entscheidenden Einfluß auf die Leistungsfähigkeit des gesamten Prozessors. Verständlicherweise kann man in einem 4-Bit-Akku nur 4-Bit-Operationen (Addition, Subtraktion usw.) durchführen. Die Verarbeitung von langen Zahlen muß dann in vielen kleinen Schritten erfolgen (hier in „Nibbles"; so nennt man 4-Bit-Gruppen). Der Vorteil von 16-Bit-Prozessoren liegt auf der Hand: Mit einem einzigen Befehl können 16-Bit-Werte, d.h. Zahlen von Null bis $2^{16} - 1 = 65535$ verarbeitet werden. Das gilt auch dann, wenn die CPU nur intern mit 16-Bit-Worten arbeitet, den externen Datentransfer aber auf einem 8-Bit-Bus vornimmt. (Natürlich verliert man dann wegen des byteweisen Datentransfers einen Teil der Geschwindigkeit gegenüber „echten" 16-Bit-CPUs.)

Manche Prozessoren gestatten es auch, bei bestimmten Operationen mehrere Register als ein einziges mit entsprechend größerer Länge zu verwenden. So kann z.B. der Z8000 vier 16-Bit-Register für zwei 32-Bit-Zahlen verwenden und legt das bis zu 64 Bit lange Ergebnis nach einem Multiplikations-Befehl in vier aufeinanderfolgenden 16-Bit-Registern ab. Bei den 8-Bit-Prozessoren sind beispielsweise die Register H und L bei 8080, 8085 und Z80 zu nennen: Sie können zusammen als 16-Bit-Zeiger verwendet werden. Daß das Fehlen von 16-Bit-Registern z.B. beim 6502 allerdings kein echter Nachteil sein muß, beweist die Besprechung der Adressierungsarten im übernächsten Kapitel.

Die Länge des Programmzählers bestimmt primär, wie groß der vom Prozessor adressierbare Speicherbereich ist. Mit 16 Bits können 2↑16 = 65536 Worte adressiert werden, bei 8-Bit-CPUs also 65536 Byte; das entspricht 65536/1024 = 64 KByte (1 KByte ist 2↑10 = 1024 Byte, nicht zu verwechseln mit 1 kByte = 1000 Byte!). Bei manchen Prozessortypen sind allerdings alle Programmzähler-Bits auch tatsächlich als Adressenbus-Anschlüsse am CPU-Gehäuse herausgeführt; dazu bringt das nächste Kapitel einige Beispiele. So ist z.B. bei der CPU 6504 der Programmzähler zwar 16 Bits lang, weil der gleiche Siliziumchip wie beim 6502 verwendet wird; der 6504 besitzt aber ein kleineres Gehäuse mit nur 13 statt 16 Adressenbus-Pins, so daß nur 2↑13 = 8192 Byte = 8 KByte adressiert werden können.

Bei den Prozessoren und Einchip-Computern mit nur 4 Bit Datenbreite fällt übrigens auf, daß der Arbeitsspeicher (für die Zwischenspeicherung von Daten) zwar nur 4 Bit breit organisiert ist, der Festwertspeicher (für die Programmbefehle) dagegen 8 Bit. Das liegt daran, daß man bei 4-Bit-ROMs ja nur 2↑4 = 16 unterschiedliche Befehle zur Verfügung hätte. Bei 8 Bit Befehlsbreite sind dagegen 2↑8 = 256 Befehle realisierbar, was für die meisten Bedürfnisse ausreicht.

Segmentierte Adressierung

Es gibt eine Reihe von Prozessoren, die einen größeren Speicher adressieren können, als es die Länge des Programmzählers eigentlich zuläßt. Dazu verwendet man die sogenannte „segmentierte" Adressierung: Ein besonderes Segmentregister schaltet den durch den Programmzähler adressierten Speicherbereich gruppenweise um. Das erschwert zwar ein wenig die Programmierung solcher CPUs, erleichtert andererseits aber auch die Verwaltung getrennter Speicherbereiche einzelner Benutzer bei Computern, an denen mehrere Benutzer gleichzeitig im Time-Sharing-Betrieb arbeiten können (Multi-User-Betrieb).

Die Segment-Adressierung ist z.B. beim 8086 bzw. bei der 8-Bit-Version 8088 realisiert. Die (physikalische) Speicheradresse ergibt sich hier aus der Summe des Programmzähler-Inhalts und des um vier Bit nach links verschobenen Segmentregister-Inhalts:

```
Segmentregister     S S S S S S S S S S S S S S S S
Programmzähler +            P P P P P P P P P P P P P P P P
Phys. Adresse     = A A A A A A A A A A A A A A A A A A A A
```

Die resultierende 20-Bit-Adresse erlaubt es dem 8086, bis zu 2↑20 = 1048576 Bytes = 1024 KByte = 1 MByte zu adressieren. Das Segmentregister adressiert den Speicher in Gruppen von 16 Bytes.

Beim Z8001 funktioniert das ganz ähnlich. Hier ist das Segmentregister um sieben Bits gegenüber dem Programmzähler (oder dem Stackpointer) versetzt. Die physikalische Adresse umfaßt also 16 + 7 Bits, damit kann man 2↑23 Bytes =

8192 KByte adressieren. Der Z8002 ist dagegen eine unsegmentierte Version der Z8000-Serie, er kann nur 64 KByte ansprechen und eignet sich daher vorwiegend für kleinere Systeme.

Vergleich üblicher Prozessoren

Die Abbildungen zu diesem Kapitel *(Abb. 1.2.2.1 bis 1.2.2.19)* zeigen die Registerstruktur der wichtigsten Prozessortypen. Einigen älteren (TMS-1000, 3870, 2650) sieht man an, daß sie noch ein wenig exotisch und eigenwillig konstruiert sind, während die neueren 8-Bit-Prozessoren 8080, Z80, 6800, 6809 und 6502 zumindest elementare Gemeinsamkeiten im Konzept erkennen lassen.

Unter den 16-Bit-Prozessoren ist der TMS-9900 nicht nur der älteste, sondern auch wohl, bedingt durch sein Workspace-Konzept, der eigenwilligste. Der 8086 erinnert mit seinem Akku-Konzept noch an seine 8-Bit-Vorläufer, während Z8000, 68000 und NS16032 als Stand der Technik gelten können.

Etwas ungewohnt ist vielleicht auch der Aufbau von Einchip-Mikrocomputern. So ist beim 8048 der Registersatz in einem besonderen, mitintegrierten RAM-Bereich untergebracht (Scratchpad Memory = Notizbuch-Speicher). Ein Teil dieses Speichers dient auch als Stack für Unterprogramm-Rücksprungadressen. Der 8035 ist eine zum Einchip-Computer 8048 kompatible „normale" CPU, d.h. sie hat kein ROM auf dem Chip.

Mit einem Scratchpad-RAM arbeiten auch die Prozessoren 3870 und F8. Sie unterscheiden sich ausschließlich in der Breite der Adressenregister (11 bzw. 16 Bit).

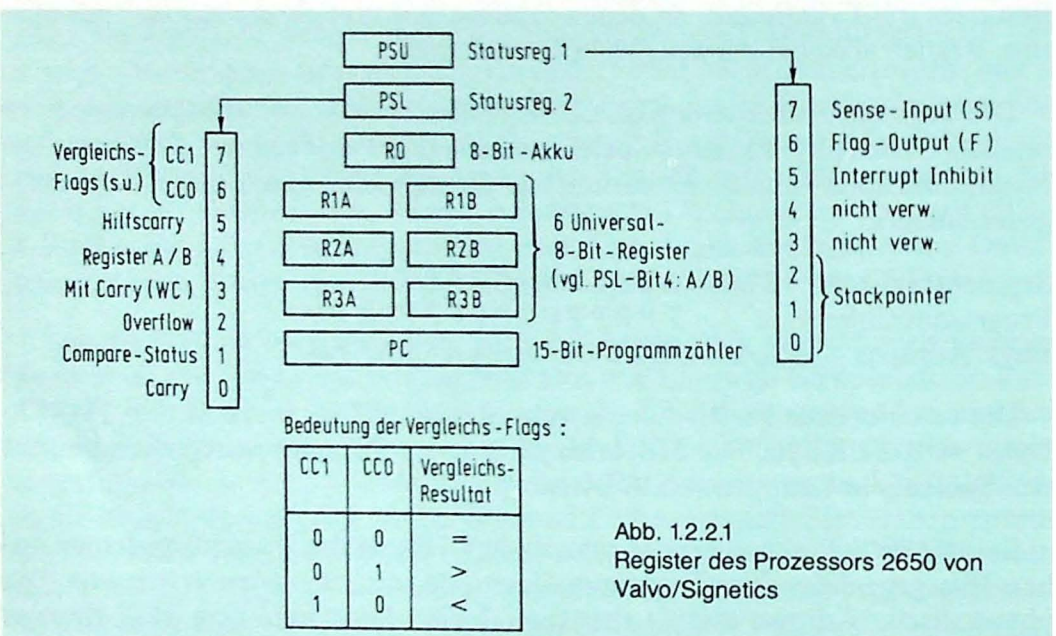


Abb. 1.2.2.1
Register des Prozessors 2650 von Valvo/Signetics

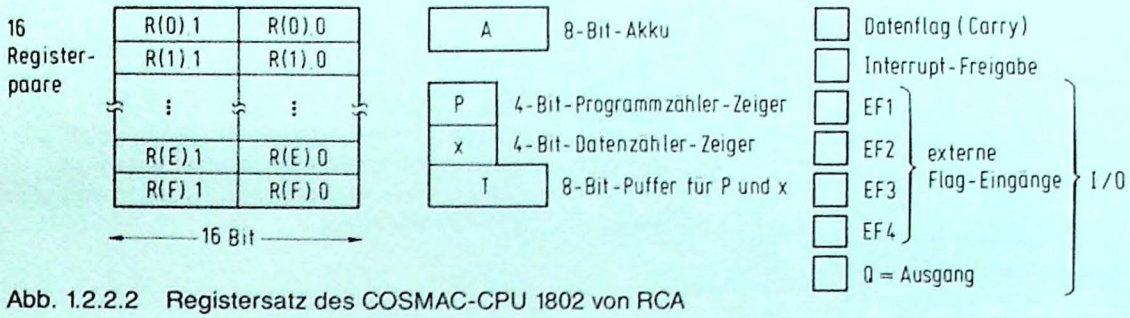


Abb. 1.2.2.2 Registersatz des COSMAC-CPU 1802 von RCA

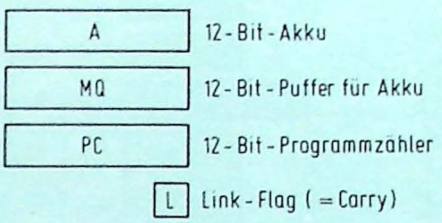


Abb. 1.2.2.3 Der 12-Bit-Prozessor IM 6100 von Intersil besitzt nur sehr wenige Register

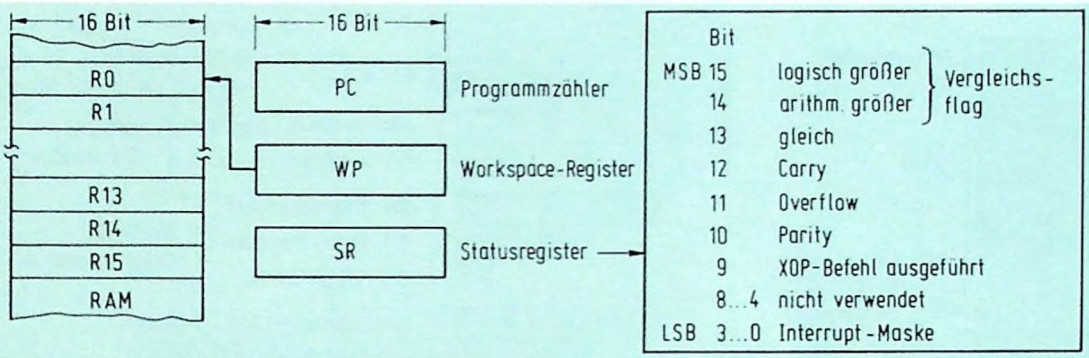


Abb. 1.2.2.4 Auf dem Chip des TMS 9900 von Texas Instruments sind nur drei Register untergebracht. Die eigentlichen Arbeitsregister liegen im RAM und werden über den Workspace-Pointer der CPU adressiert

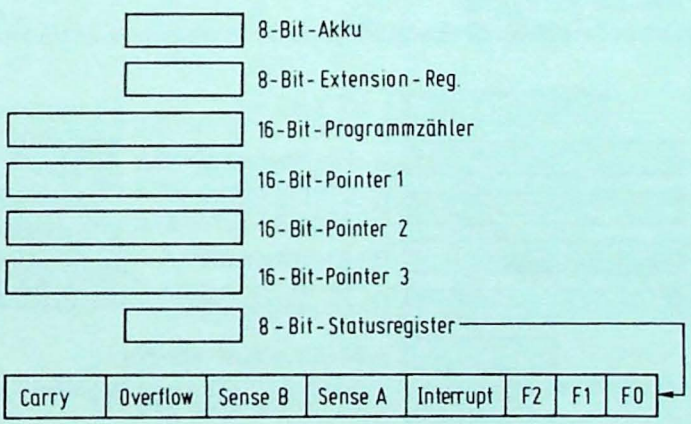


Abb. 1.2.2.5
Registersatz des SC/MP von National Semiconductor. Dieser Prozessor war einer der Pioniere, spielt heute aber kaum noch eine Rolle

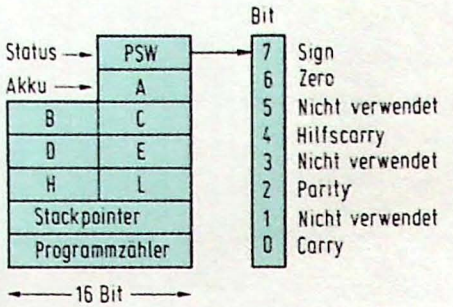


Abb. 1.2.2.6
Die CPU-Typen 8080 und 8085 von Intel gestatteten bereits das Zusammenlegen zweier Register (z. B. H und L) zu einem 16-Bit-Paar

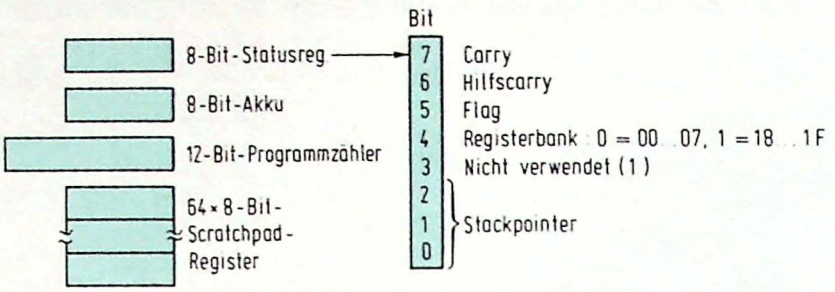


Abb. 1.2.2.7 Registersatz der Einchip-Mikrocomputer 8048, 8748 und 8035 von Intel

Abb. 1.2.2.8
Organisation des Scratchpad-RAM bei den Einchip-Computern 8048, 8748 und 8035

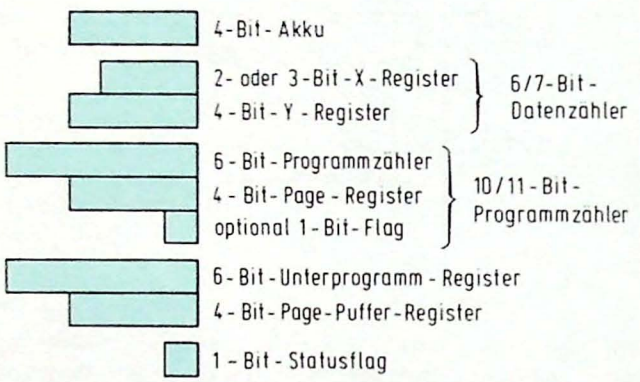


Abb. 1.2.2.9 Register im 4-Bit-Einchip-Mikrocomputer TMS-1000 von Texas Instruments

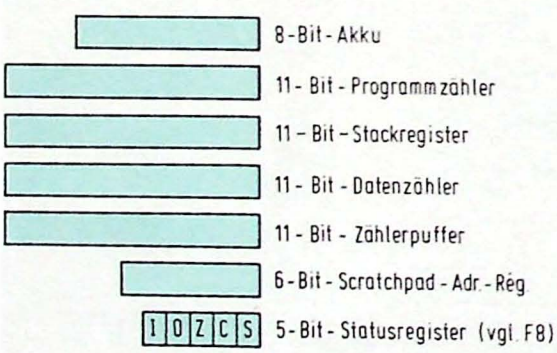


Abb. 1.2.2.10
Registersatz des Einchip-Computers 3870 von Mostek

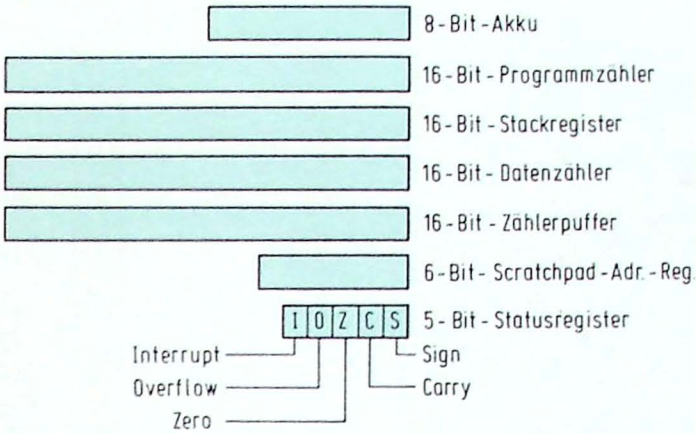


Abb. 1.2.2.11
Die Register des F8 von Fairchild unterscheiden sich nur in ihrer Länge von denen des 3870

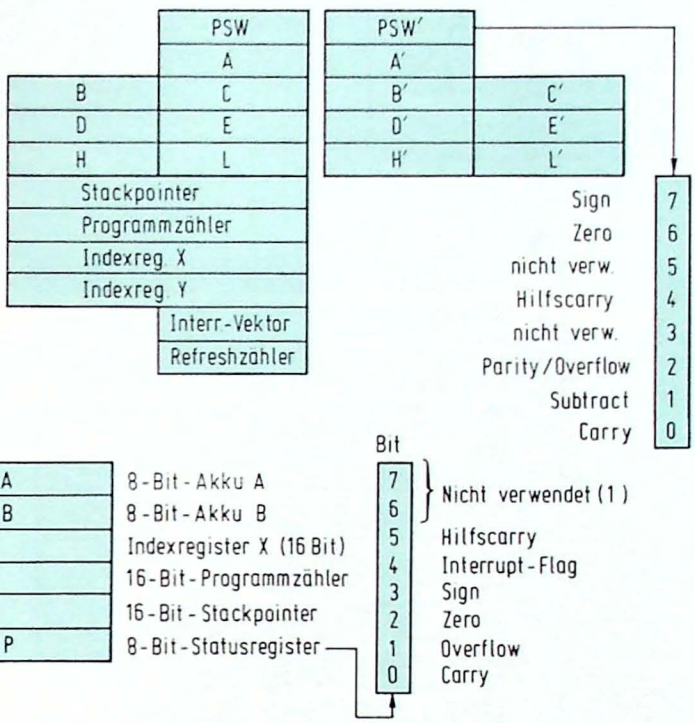


Abb. 1.2.2.12
Bei der CPU Z80 von Zilog gibt es zwei Registersätze, die z. B. bei einem Interrupt schnell vertauscht werden können, um ihren Inhalt zu retten

Abb. 1.2.2.13 Registersatz des Motorola-Prozessors 6800. Auffallend sind hier besonders die beiden Akkus

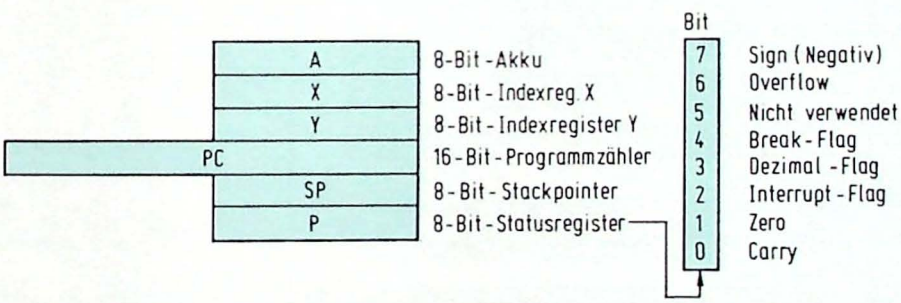


Abb. 1.2.2.14 Bei der 65XX-Familie (entwickelt von MOS Technology) ist der Programmzähler das einzige 16-Bit-Register

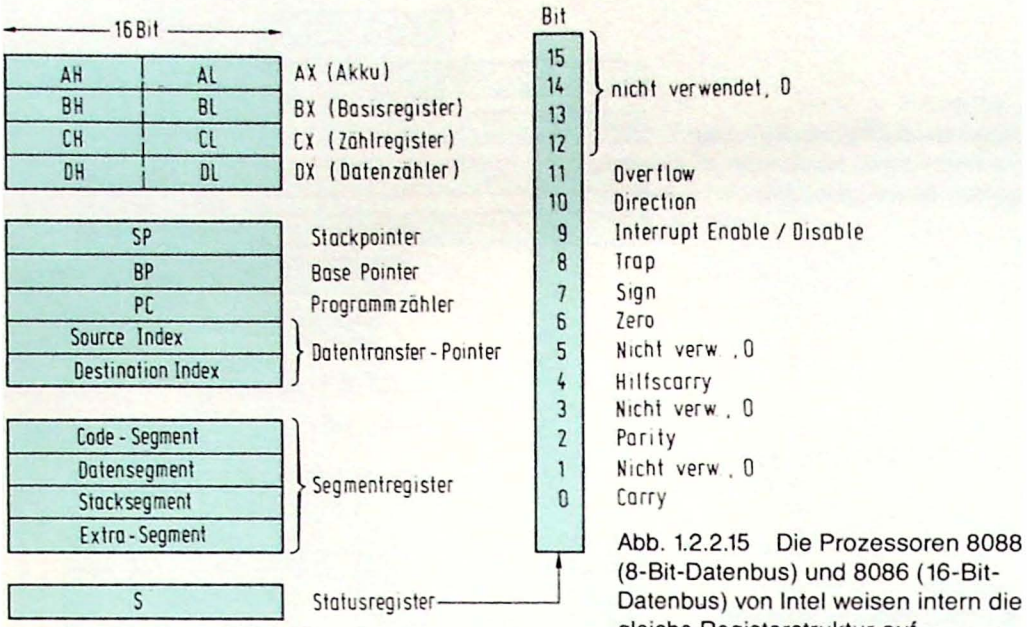


Abb. 1.2.2.15 Die Prozessoren 8088 (8-Bit-Datenbus) und 8086 (16-Bit-Datenbus) von Intel weisen intern die gleiche Registerstruktur auf

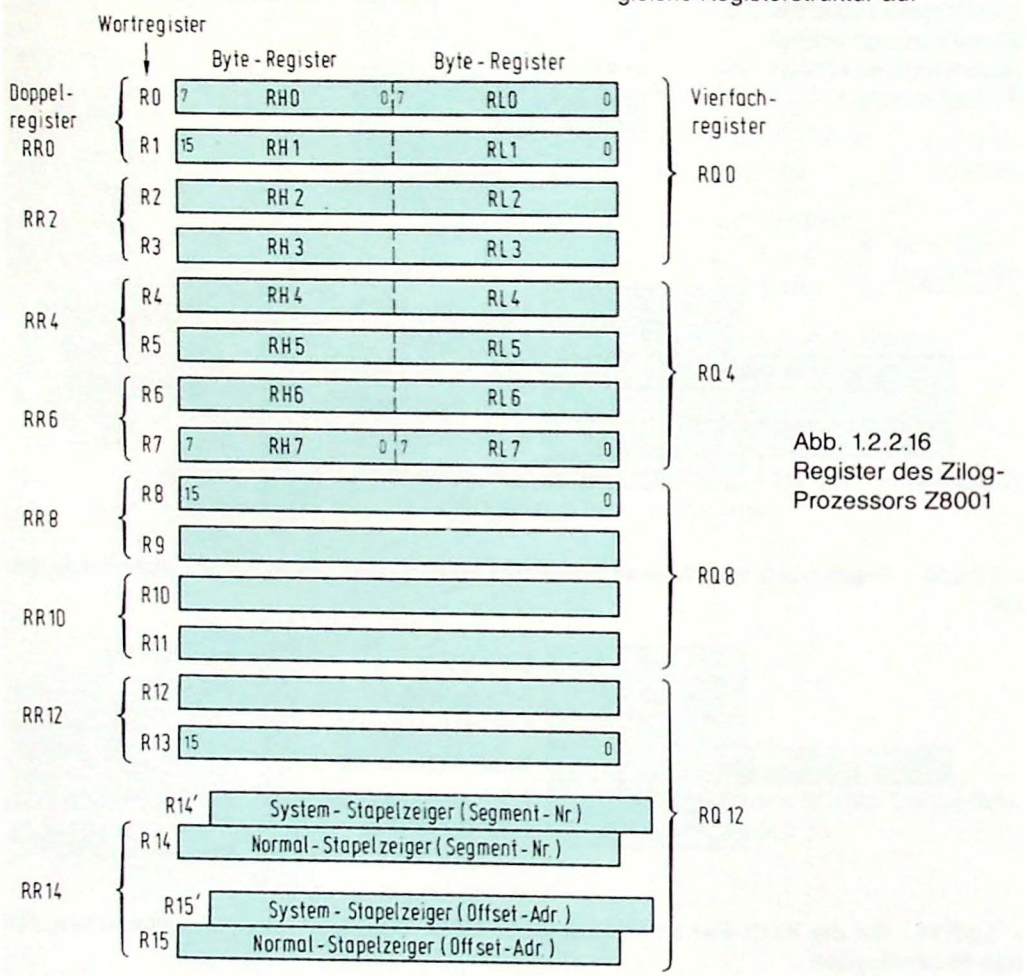


Abb. 1.2.2.16 Register des Zilog-Prozessors Z8001

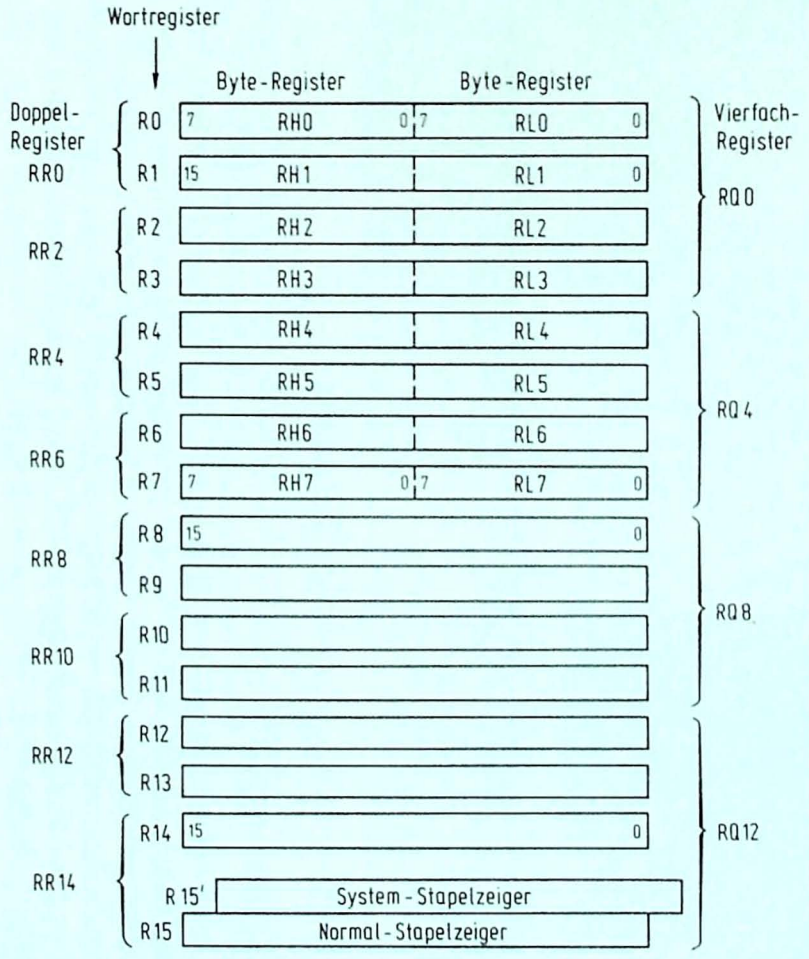


Abb. 1.2.2.17 Beim Prozessor Z8002 wurde auf segmentierte Adressierung verzichtet

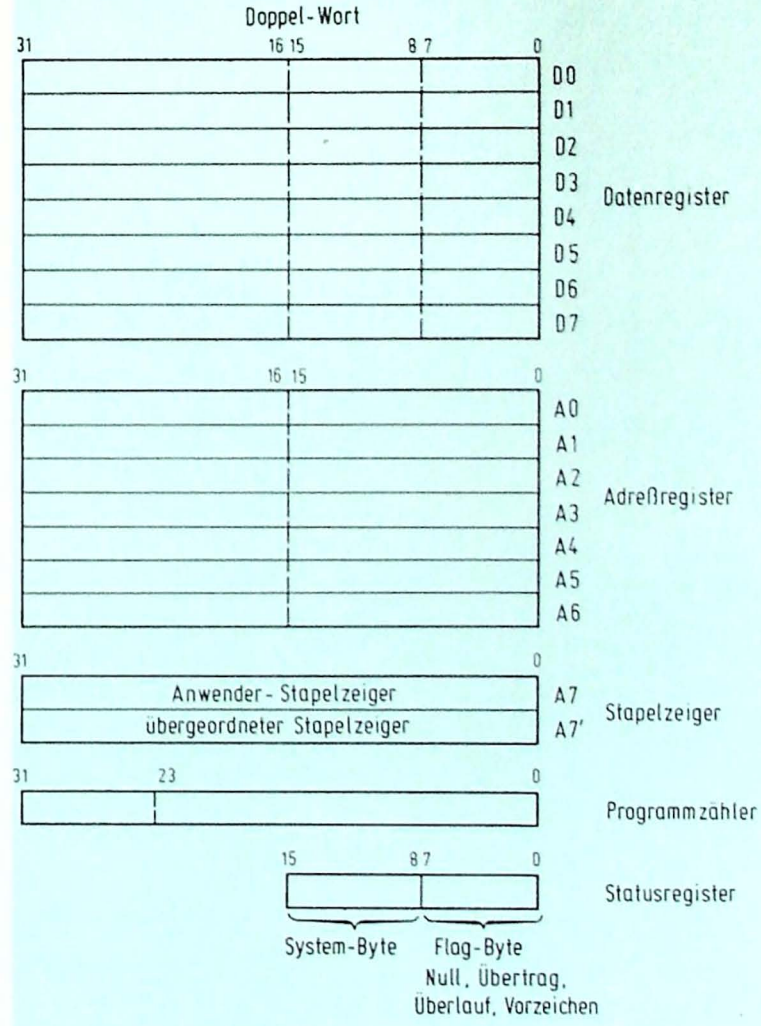


Abb. 1.2.2.18 Der Prozessor 68000 arbeitet intern mit 32 Bit Registerbreite

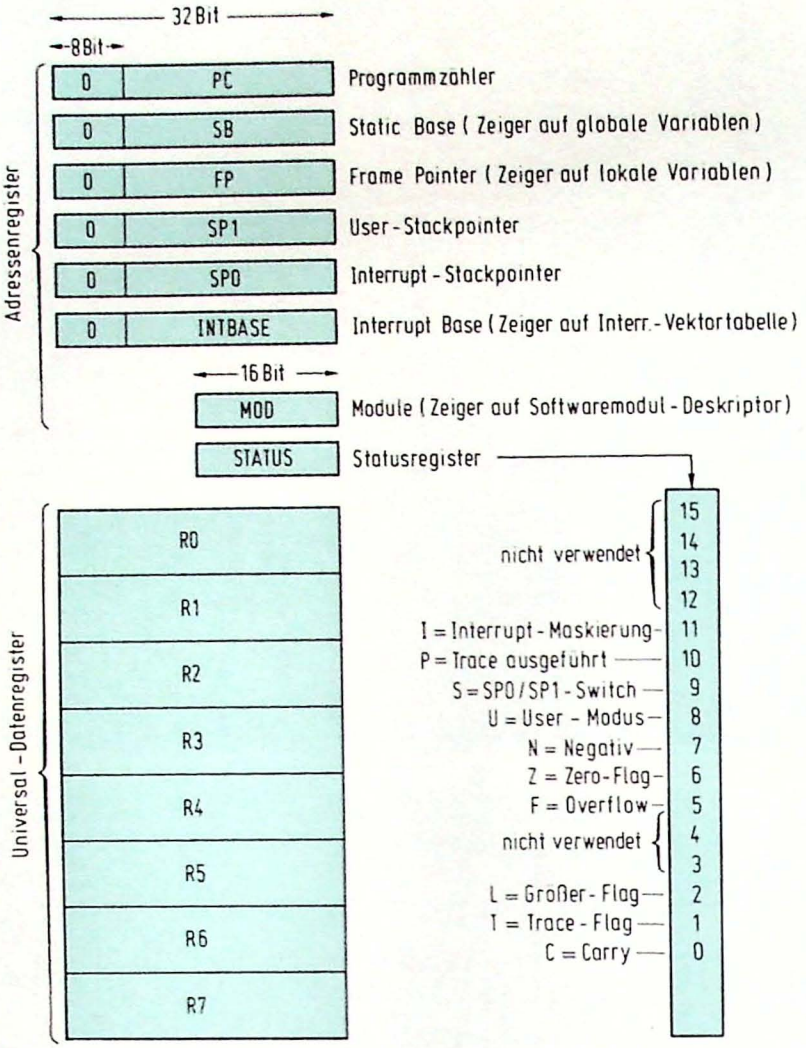


Abb. 1.2.2.19 Der Registersatz der CPU 16032 (Serie 32, National Semiconductor) ist an die Bedürfnisse von Interpretern und Compilern für höhere Programmiersprachen angelehnt

1.2.3 Gehäuseformen und Anschlußbelegung

Das Dual-In-Line-Gehäuse

Die in den Abbildungen dieses Kapitels gezeigten Anschlußbelegungen beziehen sich ausschließlich auf die heute vorwiegend verwendeten Dual-In-Line-Gehäuse (DIL) von Mikroprozessoren. Dieser Ausdruck bedeutet, daß die Anschlußbeine in zwei gegenüberliegenden Reihen entlang des Kunststoff- oder Keramik-Trägers angeordnet sind, in dem sich der eigentliche Chip, d.h. das Silizium-Substrat, befindet.

Das Gehäuse, das um ein Vielfaches größer als der Chip selbst ist, erfüllt dabei drei Aufgaben. Zum einen schützt es den empfindlichen Siliziumchip vor Umwelteinflüssen, die eine Veränderung seiner elektrischen Eigenschaften zur Folge haben könnten. Zum zweiten dient es der verbesserten Ableitung der auf dem Chip entstehenden Wärme; man muß sich vorstellen, daß bei einer Versorgungsspannung von 5 V und einer Stromaufnahme von 0,1 A (was durchaus typisch für NMOS-CPUs ist) rund 0,5 W Wärmeleistung anfallen, die irgendwie abgeführt werden müssen, weil der Chip bei Temperaturen von mehr als etwa 150 Grad Celsius den Geist aufgeben würde. Und drittens ist es natürlich nötig, die Anschlüsse des Prozessors groß genug zu machen, damit man sie bequem auf eine Platine löten kann, ohne daß die Gefahr von Kurzschlüssen oder Beschädigungen der Beine (Pins) besteht. Ihr Abstand beträgt bei DIL-Gehäusen normalerweise 2,54 mm.

Mikroprozessoren haben heute meist 28...64 Anschlüsse. Man kann sich leicht ausrechnen, daß schon ein 8-Bit-Prozessor, bei dem Adressen- und Datenbus nicht gemultiplext werden (d.h. abwechselnd auf den gleichen Pins liegen), eigentlich schon mindestens 30 Pins braucht: 16 Adressenleitungen, 8 Datenleitungen, zwei Anschlüsse für die Versorgungsspannung, Reset-Eingang, R/W-Leitung, wenigstens ein Interrupt-Eingang, Takteingang ergeben allein schon 30 Anschlüsse. Darüber hinaus stehen oft noch zusätzliche Steuerleitungen zur Verfügung. Um eine 8-Bit-CPU mit nur 28 Anschlüssen zu realisieren, ohne durch Multiplexen eine Geschwindigkeits-Verringerung in Kauf zu nehmen, gibt es deshalb nur den Weg, die Zahl der Adressenbus-Anschlüsse zu verringern, was natürlich den ansprechbaren Speicherbereich ebenfalls vermindert. Diesen Weg geht z.B. der 6504, der zwar mit dem 6502 software-kompatibel ist (d.h. die gleichen Befehle versteht), aber 28 statt 40 Gehäusepins und deshalb auch nur 13 statt 16 Adressenbusleitungen hat, womit er 8 statt 64 KByte adressieren kann. Das dadurch mögliche kleinere Gehäuse ist nicht nur in der Fertigung preisgünstiger, sondern trägt auch zu einer kleineren und damit billigeren Computer-Platine bei.

Beim 8085 dagegen war es wegen der Vielzahl der nötigen Steuerleitungen dieses Prozessors nötig, den Datenbus mit den niederwertigen acht Adressenleitungen zu multiplexen, um überhaupt mit 40 Pins auszukommen. Dies erfordert bei Speichern und Ein-/Ausgabe-Bausteinen natürlich andererseits eine Demultiplexer-Logik, um die Signale wieder zu entwirren.

Ganz besonders eklatant ist das Gehäusegrößen-Problem bei den 16-Bit-Prozessoren, bei denen ja allein der Datenbus schon 16 Pins belegen würde. Viele solcher Prozessoren multiplexen daher (ähnlich wie der 8085) den Datenbus mit dem Adressenbus (z.B. 8086, Z8000). Zum Teil wurde die interne 16- oder 32-Bit-Register-Architektur auch auf einen externen 8-Bit-Datenbus gemultiplext, der die Daten byteweise transportiert und daher nur noch 8 Leitungen benötigt (z.B. 8088, 68008). Dieses Verfahren führt zu einer noch größeren Geschwindigkeits-Einbuße als das Multiplexen der 16-Bit-Daten mit dem Adressenbus. Ehrlicherweise sollte man 8088 und 68008 daher als 8-Bit-CPUs bezeichnen. Nur wenige

16-Bit-CPUs führen Adressen- und Datenbus in voller Breite ohne Multiplexen heraus (z.B. TMS-9900, 68000) und benötigen dafür ein relativ teures Gehäuse mit 64 Anschlußbeinen.

Bei den DIL-Gehäusen ist die Seite, auf der Pin 1 liegt, gewöhnlich mit einer Nase oder einem eingeprägten Punkt markiert. Die Zählweise erfolgt bei Draufsicht (Platinen-Bestückungsseite) gegen den Uhrzeigersinn bzw. bei Blick von unten (Lötseite von Platinen) im Uhrzeigersinn. Die *Abbildungen* dieses Kapitels zeigen die Anschlußbelegungen bei Draufsicht.

Für hohe Anforderungen an die Beständigkeit gegenüber Umwelteinflüssen (Temperatur, Feuchtigkeit) verwendet man als Gehäusematerial meist Keramik. Bei durchschnittlichen Anforderungen wird dagegen ein gewöhnlich schwarz eingefärbter Kunststoff verwendet. Auf der Oberseite des Gehäuses ist ein Kurzzeichen für die Herstellerfirma, die Typenbezeichnung des Prozessors sowie eine Serienkennzeichnung aufgedruckt.

Andere Gehäuseformen

Wie schon erwähnt, ist das DIL-Gehäuse am weitesten verbreitet. Für spezielle Anwendungen gibt es auch die Gehäuseformen Single-In-Line (nur eine einzige Pinreihe; das Gehäuse wird dann meist senkrecht auf die Platine gelötet) oder, bei CPUs mit sehr vielen Anschlußbeinen, das Quad-In-Line-Gehäuse, das Pins auf jeder der vier Gehäusekanten besitzt.

Bei allen vorgenannten Gehäusen werden die Anschlußpins vor dem Verlöten durch entsprechende Löcher in der Platine gesteckt. Bei den sogenannten Flat-Pack-Gehäusen ist das nicht der Fall; sie werden direkt auf die Platinen-Leiterbahnen aufgelötet. Dann gibt es noch die Möglichkeit, Prozessoren (oder andere integrierte Schaltungen) direkt als Siliziumchip zusammen mit anderen Bauelementen auf ein Träger-Substrat (z.B. Keramik oder Glas) zu kleben. Die Verbindungen werden dann z.B. mit dünnen Golddrähten hergestellt, wonach das komplette Modul in Kunststoff vergossen wird (Dick- und Dünnfilmtechnik).

Pin-Bezeichnungen

Die Anschlüsse an Mikroprozessoren sind von den Herstellern nicht ganz einheitlich gekennzeichnet. Die folgende Aufstellung gibt die nötigen Erläuterungen zu den Anschlußbildern *(Abb. 1.2.3)*. n steht dabei für irgendeine Zahl oder Nummer. In den Abbildungen bedeutet ein Strich über einer Bezeichnung, daß diese Leitung „aktiv low" ist. Beispielsweise heißt $\overline{\text{RESET}}$, daß diese Leitung normalerweise high (1) ist und dann, wenn ein Reset ausgelöst werden soll, von außen kurz auf

Low-Pegel gezogen wird. In manchen Datenblättern wird diese Invertierung der logischen Zuordnung statt einem darüberstehenden Strich durch eine Linie hinter der Bezeichnung gekennzeichnet. Der mit aufgeführte Baustein 80287 ist übrigens ein Arithmetik-Prozessor.

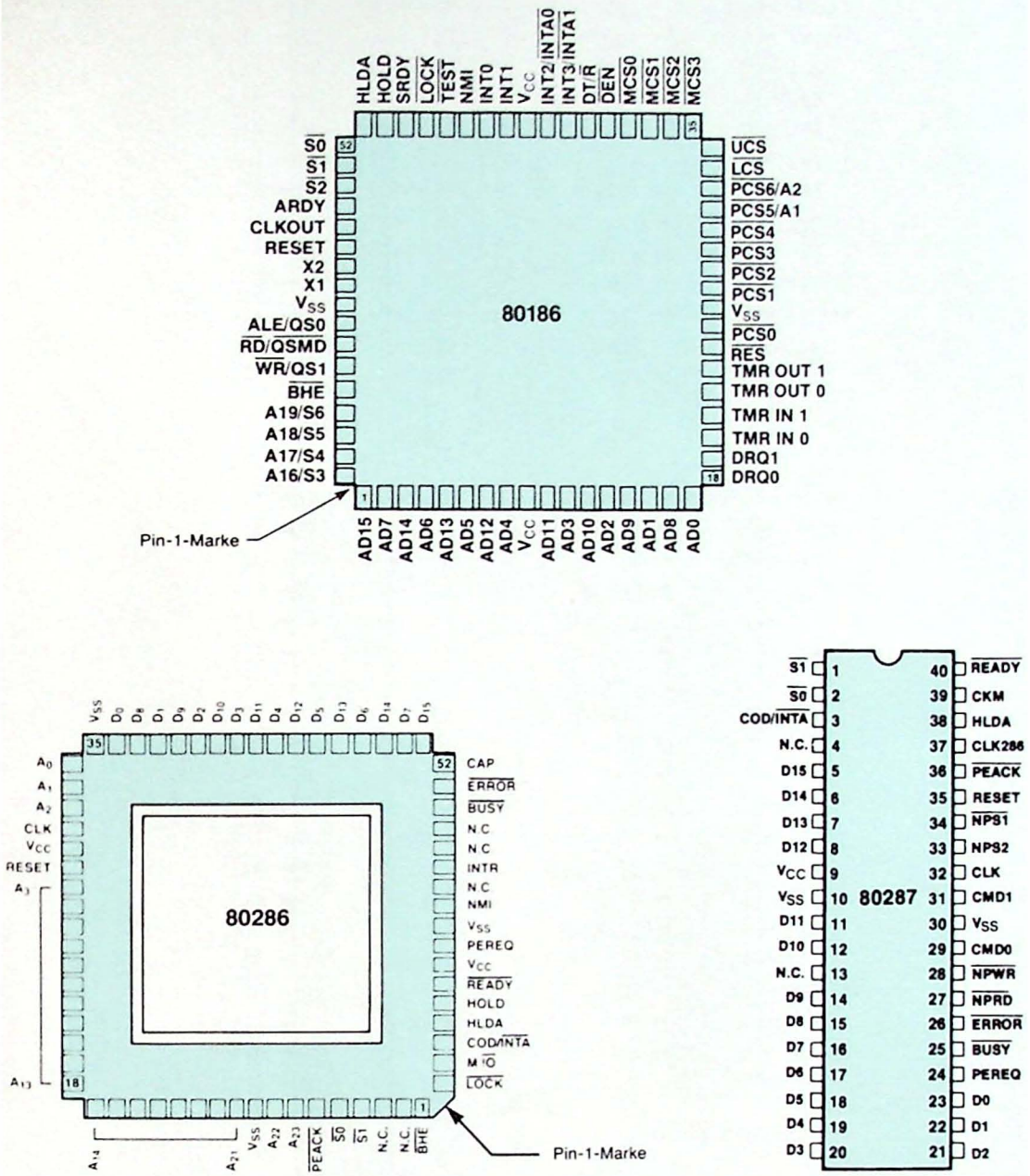
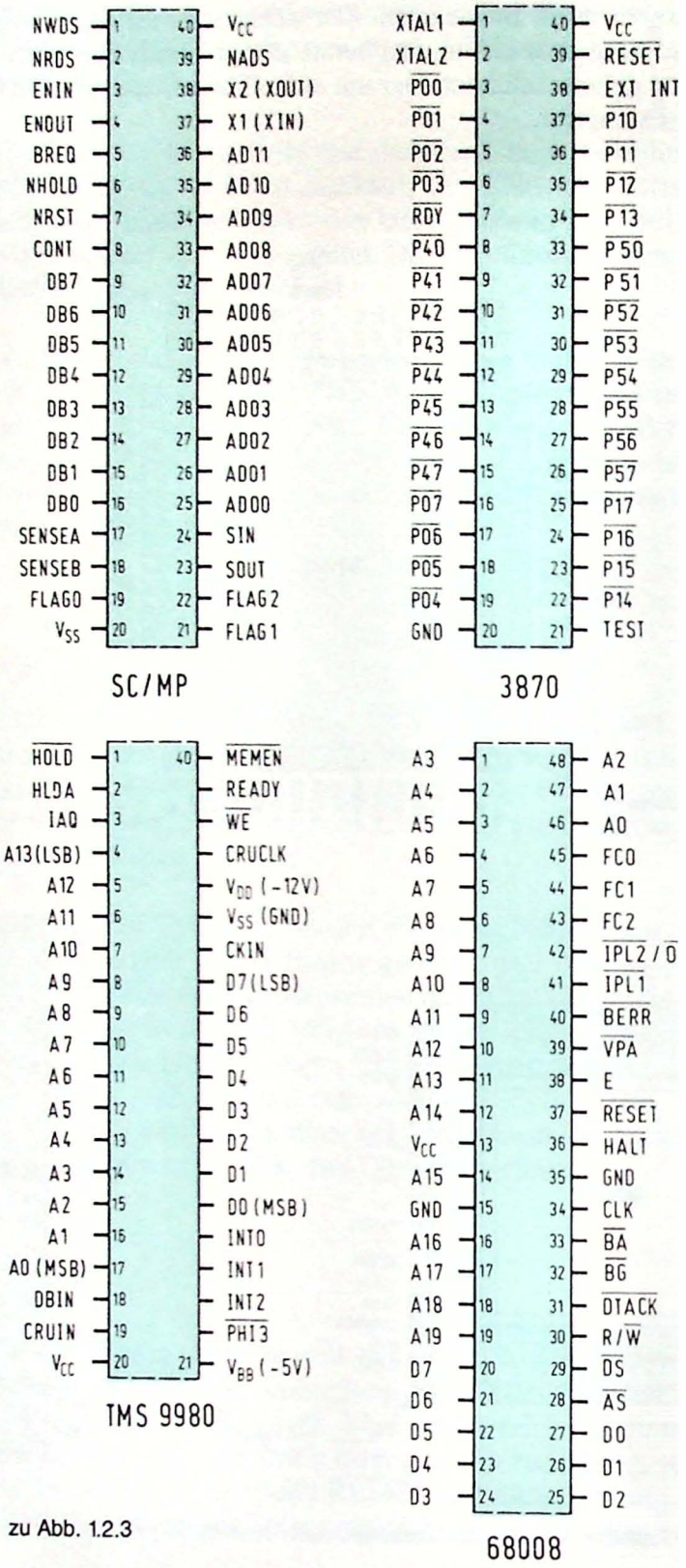
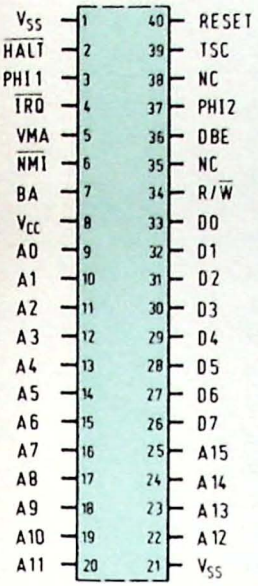


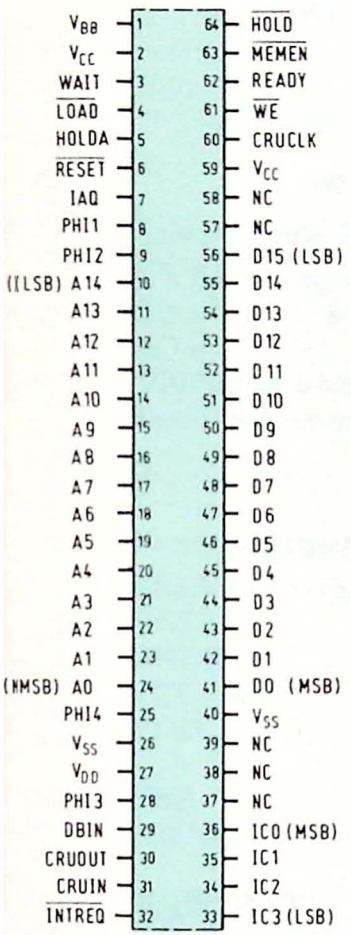
Abb. 1.2.3 Anschlußbilder von Mikroprozessoren; außer beim 80186 jeweils von oben gesehen


NWDS 1 40 V_{CC}
NRDS 2 39 NADS
ENIN 3 38 X2 (XOUT)
ENOUT 4 37 X1 (XIN)
BREQ 5 36 AD11
NHOLD 6 35 AD10
NRST 7 34 AD09
CONT 8 33 AD08
DB7 9 32 AD07
DB6 10 31 AD06
DB5 11 30 AD05
DB4 12 29 AD04
DB3 13 28 AD03
DB2 14 27 AD02
DB1 15 26 AD01
DB0 16 25 AD00
SENSEA 17 24 SIN
SENSEB 18 23 SOUT
FLAG0 19 22 FLAG2
V_{SS} 20 21 FLAG1
SC/MP
XTAL1 1 40 V_{CC}
XTAL2 2 39 RESET
P00 3 38 EXT INT
P01 4 37 P10
P02 5 36 P11
P03 6 35 P12
RDY 7 34 P13
P40 8 33 P50
P41 9 32 P51
P42 10 31 P52
P43 11 30 P53
P44 12 29 P54
P45 13 28 P55
P46 14 27 P56
P47 15 26 P57
P07 16 25 P17
P06 17 24 P16
P05 18 23 P15
P04 19 22 P14
GND 20 21 TEST
3870
HOLD 1 40 MEMEN
HLDA 2 READY
IAQ 3 WE
A13(LSB) 4 CRUCLK
A12 5 V_{DD} (-12V)
A11 6 V_{SS} (GND)
A10 7 CKIN
A9 8 D7(LSB)
A8 9 D6
A7 10 D5
A6 11 D4
A5 12 D3
A4 13 D2
A3 14 D1
A2 15 D0 (MSB)
A1 16 INT0
A0 (MSB) 17 INT1
DBIN 18 INT2
CRUIN 19 PHI3
V_{CC} 20 21 V_{BB} (-5V)
TMS 9980
A3 1 48 A2
A4 2 47 A1
A5 3 46 A0
A6 4 45 FC0
A7 5 44 FC1
A8 6 43 FC2
A9 7 42 IPL2 / 0
A10 8 41 IPL1
A11 9 40 BERR
A12 10 39 VPA
A13 11 38 E
A14 12 37 RESET
V_{CC} 13 36 HALT
A15 14 35 GND
GND 15 34 CLK
A16 16 33 BA
A17 17 32 BG
A18 18 31 DTACK
A19 19 30 R/W
D7 20 29 DS
D6 21 28 AS
D5 22 27 D0
D4 23 26 D1
D3 24 25 D2
68008

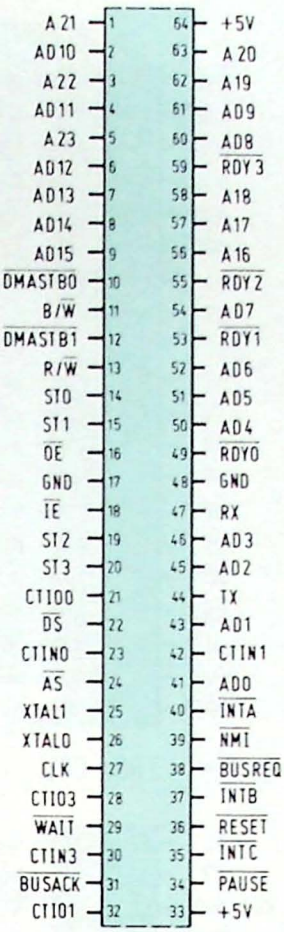
zu Abb. 1.2.3



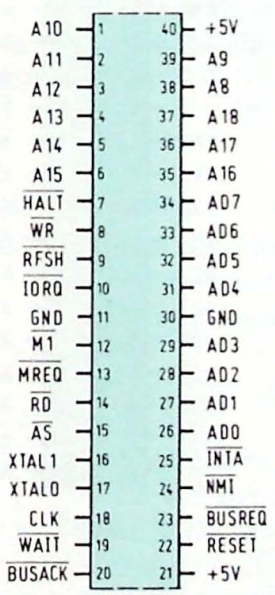
MC 6800



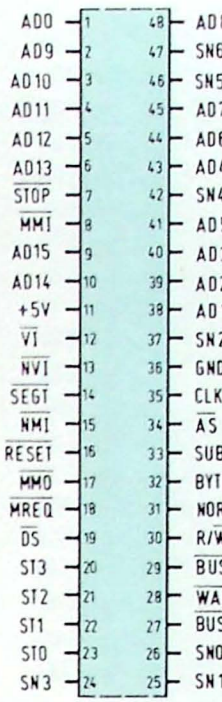
TMS 9900



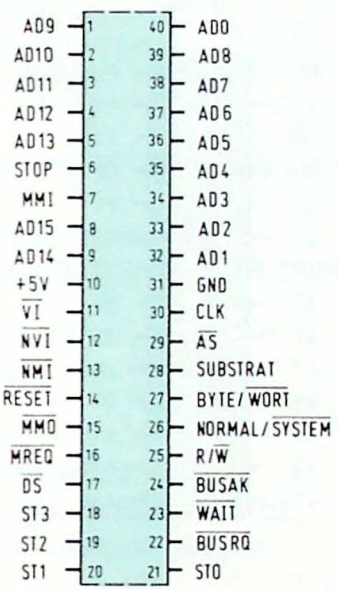
Z 8216 MPU
(Z 800 - Fam.)



Z 8108 MPU
(Z 800 - Fam.)

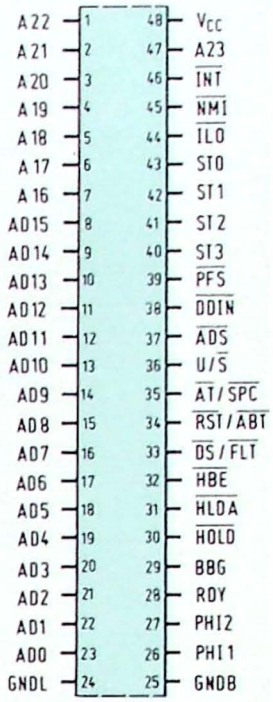


Z 8001

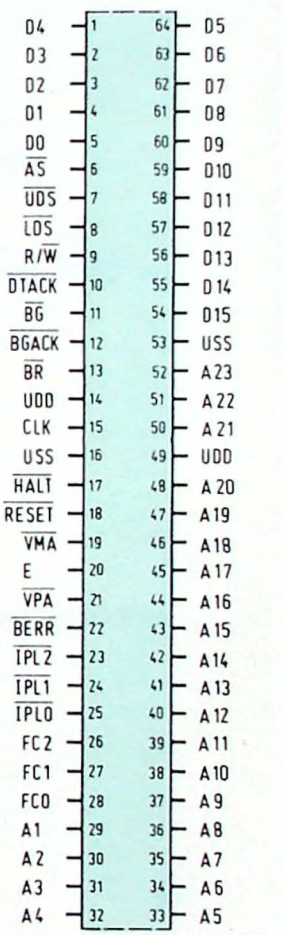


Z 8002

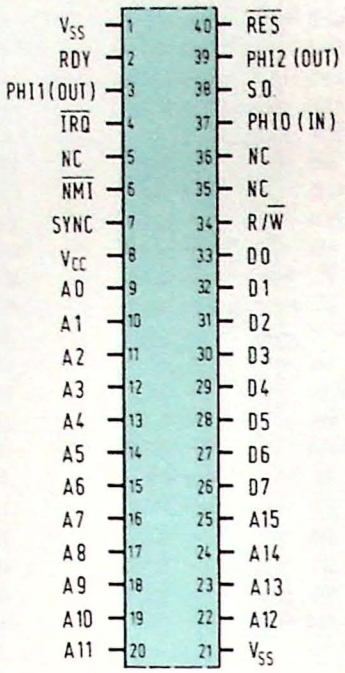
zu Abb. 1.2.3



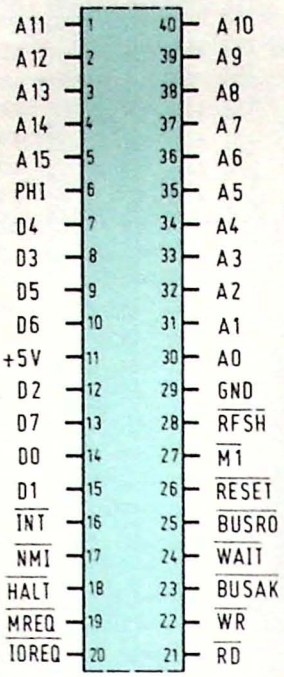
NS 16032



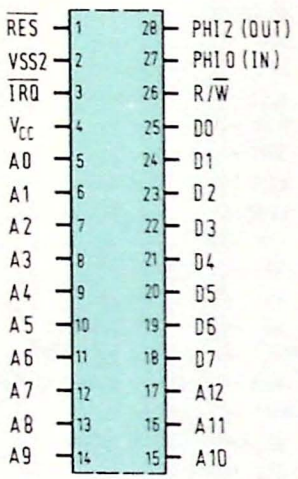
68000



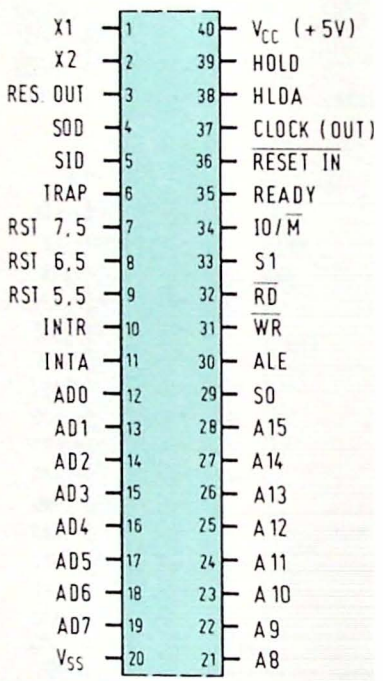
6502



Z80 – CPU

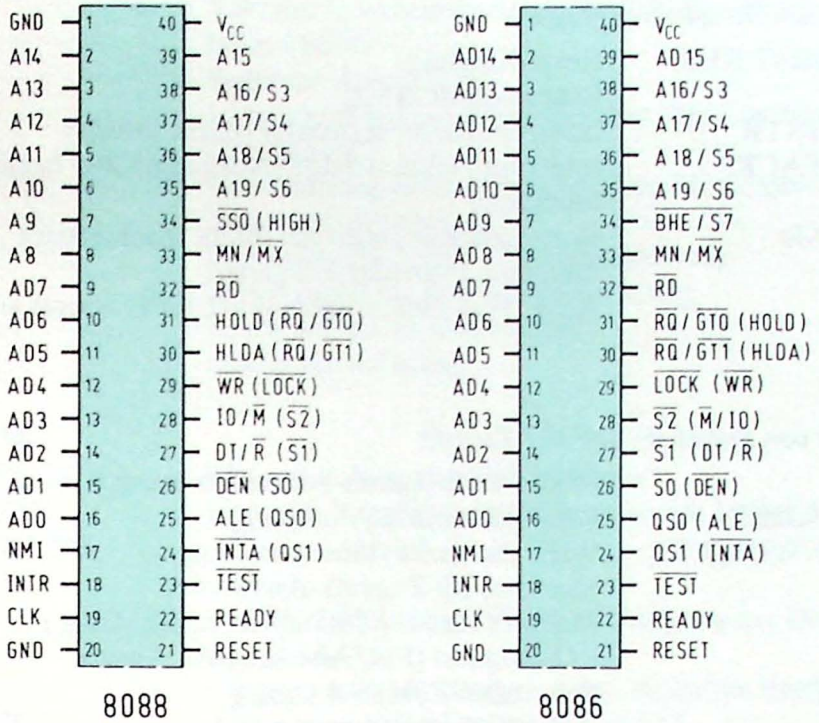


6504



8085

zu Abb. 1.2.3



zu Abb. 1.2.3

Adressen- und Datenbus:

A n,AB n	Adressenbus (A0 = LDS/UDS beim 68000!)
D n,DB n	Datenbus
AD n	Adressen und Daten gemultiplext

(Zählweise normal: n = 0 niederwertigstes Bit (LSB); bei TMS-9900/9980 von Texas Instruments umgekehrt!)

Versorgungsspannungs-Anschlüsse:

USS,VSS,GND	Masse (Bezugspotential, 0 V)
GNDL	Logic Ground, Logik-Bezugspotential
GNDB	Battery Ground, Versorgungs-Masse
VCC,UCC	+ 5 V
VDD, UDD	+ 5 V, beim 8080 + 12 V, + 25 V zum Programmieren beim 8048
VBB	− 5 V beim 8080
PROG	+ 25 V Programmierspannung f. EPROM bei Einchip-Computern
SUBSTRAT	Bezugspotential des Chips
N.C.	Not connected = nicht angeschlossen

Interrupt- und Reset-Anschlüsse:

RESET,NRST,RES	Reset-Eingang
RST	Restart (ähnl. NMI)
IRQ,INT,INTR	Interrupt Request (maskierbarer Interr.)
INTA,INTACK	Interrupt Acknowledge (Ausgang, CPU bestätigt Interrupt)
NMI,LOAD	Non-maskable Interrupt (nicht maskierbar)
IP n,IC n	Interrupt-Prioritäts-Eingänge
HALT	Ausg.: Halt-Befehl ausgeführt, CPU wartet auf Interrupt

Steuerung von Speicher- und I/O-Zugriff:

R/W	Read/Write (Lesen/Schreiben-Ausg.)
RD,NRDS,DBIN	Read (Lesezyklus-Ausgang)
WR,WE,NWDS,WRP	Write (Schreibzyklus-Ausgang)
M/IO	Memory/IO-Zugriffs-Auswahl
MREQ	Memory Request (Speicherzugriff, Ausg.)
IORQ	I/O-Request (Ein/Aus.-Zugriff, Ausg.)
M1,SYNC	Befehlshol-Zyklus-Ausgang
RFSH	Refresh-Zyklus-Ausgang
DBE,DBUSEN	Data Bus Enable (Eingang für Datenbus-Freigabe in Verbindung mit TSC)
DMA,ENIN	Direct Memory Access (Eingang); CPU trennt sich vom Bus
TSC,ADREN	Three state control; wie DMA, jedoch nur für Adressenbus
BUSRQ,BR	Bus request; wie DMA
BUSAK,ENOUT	Bus acknowledge (Ausg.); CPU
BGACK,BG	bestätigt DMA/BUSRQ
BA	Bus available (Bus verfügbar; ähnlich BUSAK)
HOLD	wie DMA; Busabtrenn-Eing.
HOLDA,HLDA	wie BUSAK
LOCK	Eingang für Sperren von HOLD
VMA,AS	Valid memory address (Ausgang); gültige Adresse liegt an
VPA	Valid peripheral address (Ausgang); gültige Peripherie-Adresse liegt an
MEMEN	Memory enable; wie VMA
BHE,UDS	Umschalten auf höherwertiges Byte bei 16-Bit-CPUs
LDS	Umschalten auf niederw. Byte (16-Bit-CPUs)
QS	Queue status = Status des Befehls-Zwischenspeichers
ALE,NADS	Address latch enable (Adresse liegt an bei gemultiplextem Bus)
EA	Externer Speicherzugriff bei Einchip-Computern

E	Takt zur Synchronisation mit 6800-Peripherie beim 68000
S	Segment-Auswahl
PSEN	Lesezugriff auf externen Speicher (bei Einchip-Computern)
MN/MX	Umschaltung beim 8086 zwischen Minimum- und Maximum-System
RQ,GT	Busprioritäts-Steuerung (bidirektional)
DT/R	Steuersignal für Bustreiber-Richtung
DEN	Latch-Signal für Bustreiber
BERR	Busfehler-Ausgang

Zeitsteuerung:

PHI,CLK,CLOCK	Ein- oder Ausgang für Taktsignal
READY,RDY, NHOLD,PAUSE, DTACK	Eing. für Verzögerung bei langsamem Speichern
WAIT	wie READY; beim 9900 aber Ausgang zur Bestätigung von READY
CONT	Eingang zum vorübergehenden Anhalten der CPU
RUN/WAIT	Ausgang zur Bestätigung von WAIT
X n,XTAL n	Anschlüsse für Schwingquarz
S n,FC n	Ausg. für CPU-Zustandsanzeige

Ein/Ausgabeleitungen:

SO	Set Overflow Flag (Eingang, 6502)
FLAG n	Statusflag-Ausgänge (SC/MP)
SIN,SID,CRUIN	Serial data in (serieller Eing.)
SOUT,SOD,CRUOUT	Serial data out (ser. Ausgang)
SENSE n	Flag-Eingang
T n,TEST	Testeingang (beim 8048 auch Zähleingang)
P n,PA n,PB n	I/O-Port-Leitung (bei Einchip-Computern)

1.2.4 Elektrische Eigenschaften

Das nun folgende Kapitel befaßt sich mit den zulässigen Spannungen (d.h. den Pegeln, die den logischen Werten 0 und 1 zugeordnet sind) und mit dem Zeitverhalten von Mikroprozessoren (Taktfrequenz, Speicher-Zugriffszeit usw.). Die Hersteller garantieren die Funktionsfähigkeit ihrer Produkte abgesehen von der Einhaltung dieser Werte allerdings nur in bestimmten Umgebungstemperatur-Bereichen. Für normale Anforderungen genügt eine Spezifikation von 0...70 Grad Celsius;

ein größerer Bereich bedeutet meist automatisch einen höheren Preis für das Produkt. Erwähnt werden muß noch, daß bei 70 Grad Celsius Umgebungstemperatur der Chip selbst eine wesentlich höhere Temperatur erreicht (über 120 Grad), da ja die Wärmeleitfähigkeit des Kunststoff- oder Keramikgehäuses zur umgebenden Luft nicht ideal sein kann. Bei CMOS-Prozessoren, die von Haus aus weit weniger Leistung benötigen als NMOS-CPUs, tritt dieses Problem aber wesentlich weniger in Erscheinung.

Pegel-Definition

NMOS- und CMOS-Prozessoren werden überwiegend mit einer einzigen Versorgungsspannung von +5 V betrieben; nur ältere CPUs wie der 8080 benötigen noch zusätzliche Spannungen (hier −5 V und +12 V). Die Hersteller lassen üblicherweise eine Toleranz von +/−5% zu; die 5 V dürfen also zwischen 4,75 V und 5,25 V schwanken. Außerhalb dieses Bereichs ist eine einwandfreie Funktion nicht gewährleistet. Eine bleibende Beschädigung ist jedoch erst bei Überschreiten von etwa 7 V oder falscher Polarität zu befürchten. Bei CMOS-CPUs ist die Versorgungsspannungs-Toleranz meist größer (z.B. +/−10%, d.h. 4,5...5,5 V).

Auch der Strombedarf ist technologieabhängig. Hier der Verbrauch einiger NMOS-CPUs bei 5 V Spannung:

2650 150 mA max.
8085 170 mA max.
Z80 150 mA max.
6800 100 mA typ., 200 mA max.
6502 50 mA typ., 120 mA max.
8086 275 mA max.

Im Gegensatz dazu ist bei CMOS-Prozessoren der Stromverbrauch nicht konstant, sondern von der Taktfrequenz abhängig. CMOS bedeutet ja „complementary metaloxide/silicon": Jeweils zwei Feldeffekt-Transistoren sind in Serie geschaltet, wovon immer nur einer leitet (siehe Kapitel „CMOS-Schaltungen"). Im Ruhezustand (statisch) fließt daher so gut wie gar kein Strom, von Isolationsverlusten einmal abgesehen. Erst dann, wenn die Transistoren schalten, fließt jeweils kurzzeitig ein Strom zum Umladen der unvermeidbaren Kapazitäten auf dem Chip und auf den Leiterbahnen der Platine des Computers. Der Stromverbrauch nimmt damit proportional zur Zahl der Schaltvorgänge pro Zeiteinheit und damit auch proportional zur Taktfrequenz zu.

So beträgt der Stromverbrauch der CMOS-CPU 65C02 bei 1 MHz Taktfrequenz 4 mA, bei 2 MHz 8 mA. Hält man den Takt vorübergehend an, so wird der Verbrauch unmeßbar klein.

Das Anhalten des Taktes ist übrigens keineswegs bei allen CPU-Typen uneingeschränkt zulässig. Manche NMOS-CPUs (z.B. 6502) verwenden nämlich dynamische Speicherzellen für ihre Register (siehe Kapitel über Schreib-Lese-Speicher),

bei denen die bei CMOS-CPUs eher störenden (weil stromfressenden) Kapazitäten zur Informationsspeicherung mißbraucht werden. Hält man nun den Takt für mehr als einige tausendstel Sekunden (ms) an, so fließt die Ladung über nicht ideal isolierende Sperrschichten ab, bevor sie der Prozessor intern neu auffrischen kann, und die Information in den Registern ist verloren. Der Vorteil der dynamischen Speicherzellen ist auf der anderen Seite, daß sie weniger Chipfläche und weniger Transistoren pro Speicherzelle benötigen und auch weniger Strom verbrauchen als statisch aufgebaute NMOS-Zellen.

Zu den elektrischen Eigenschaften gehört auch die eindeutige Zuordnung von bestimmten Spannungen zu den logischen Werten 0 und 1. Hierzu gibt es rein prinzipiell schon zwei Möglichkeiten:

Positive Logik: hohe Spannung (high) = 1
niedrige Spannung (low) = 0

Negative Logik: hohe Spannung (high) = 0
niedrige Spannung (low) = 1

Tatsächlich werden beide Arten bei Mikrocomputern verwendet. Eine Leitung mit negativer Logik wird aber mit einem Strich besonders gekennzeichnet, der entweder über oder neben dem Namen steht.

Beispiel für positive Logik:

R Read-Signal (lesen). Der Speicher wird ausgelesen, wenn R auf High-Pegel liegt.

Beispiel für negative Logik:

$\overline{W}$ Write-Signal (schreiben). In den Speicher wird geschrieben, wenn diese Leitung auf Low-Pegel liegt.

Bei vielen CPUs sind diese beiden Signale zu einer einzigen Leitung zusammengefaßt:

R/$\overline{W}$ Gelesen wird bei High-Pegel (Read), geschrieben bei Low-Pegel (Write).

Die Frage ist nun noch, was ist high (hohe Spannung) und low (niedrige Spannung). Sinnvollerweise verwendet man für den High-Pegel keine höhere Spannung als die Betriebsspannung und für Low eine Spannung von nahezu Null. In der Praxis liegt die Entscheidungsschwelle zwischen High und Low nicht genau in der Mitte, also bei der halben Betriebsspannung, sondern technologiebedingt etwas tiefer. Außerdem läßt man in der Mitte einen gewissen „verbotenen" Spannungsbereich zur Sicherheit frei, bei dem die Entscheidung zwischen 0 und 1 nicht eindeutig wäre. Hier eine Übersicht der Zuordnung bei unterschiedlichen CPUs:

	NMOS 8085	NMOS Z80	NMOS 6502	CMOS 65C02
Low	<0,8 V	<0,8 V	<0,8 V	<0,8 V
High	>2,0 V	>2,0 V	>2,4 V	>2,4 V

Man beachte, daß die CMOS-CPU hier für die gleichen Werte spezifiziert ist wie die NMOS-Version. Dies ist nicht selbstverständlich, wie wir bei der Besprechung der CMOS-Schaltungen in einem späteren Kapitel noch sehen werden; der Hersteller hat dies absichtlich durch technologische Maßnahmen erreicht, um die Austauschbarkeit von NMOS- und CMOS-Version zu gewährleisten.

Die Spannung an irgendeinem Anschluß des Prozessors darf üblicherweise nicht kleiner als −0,3 V sein und die Betriebsspannung nicht um mehr als 0,3 V überschreiten, da sonst Beschädigungen der integrierten Logik auftreten können. Damit das verhindert wird, besitzen die meisten NMOS- und CMOS-ICs integrierte Schutzdioden an ihren Eingängen, die in Sperrichtung nach Masse (Bezugspotential, null Volt) und zur Versorgungsspannung geschaltet sind. Sie schützen den Chip auch gegen die hohen Spannungen, die auftreten können, wenn man sich auf einem Kunststoff-Teppich statisch aufgeladen hat und nun den Prozessor mit den Fingern berührt. Dieser Schutz ist aber nur unvollkommen, denn bei sehr hohen Spannungen sind die Dioden nicht schnell genug, um rechtzeitig vom Sperrzustand auf Durchlaß umzuschalten. Trotz der Schutzdioden sollte man also statische Entladungen an NMOS- oder CMOS-Bauelementen vermeiden.

Zeitverhalten

Neben dem statischen Verhalten spielt auch das Zeitverhalten von Prozessoren eine wichtige Rolle, die sogenannten dynamischen Eigenschaften.

Dazu gehören im wesentlichen die Taktfrequenz, mit der die CPU betrieben werden darf, und die zeitliche Abfolge der unterschiedlichen Steuersignale im Zusammenwirken mit dem Daten- und Adressenbus.

Die Taktfrequenz wählt man in der Regel möglichst hoch, da sie direkt die Arbeitsgeschwindigkeit der CPU und somit des gesamten Computers bestimmt. Ein direkter Vergleich von unterschiedlichen Prozessortypen allein aufgrund ihrer maximalen Taktfrequenz ist allerdings nicht sinnvoll, da je nach interner Decodierung der Befehle mehr oder weniger Taktzyklen (Periodendauern der Taktfrequenz) für einander entsprechende Befehle benötigt werden. Die folgende Aufstellung macht dies deutlich. Sie vergleicht die benötigten Zeiten einiger 8-Bit-CPUs für die Ausführung eines Befehls, der den Inhalt einer beliebigen Speicherzelle in den CPU-Akku lädt.

(Die Werte von 16-Bit-CPUs sind hier absichtlich nicht mit aufgeführt, da sie gewöhnlich mit mehr Adressen- und Datenbits arbeiten und die Ergebnisse somit nicht vergleichbar wären.)

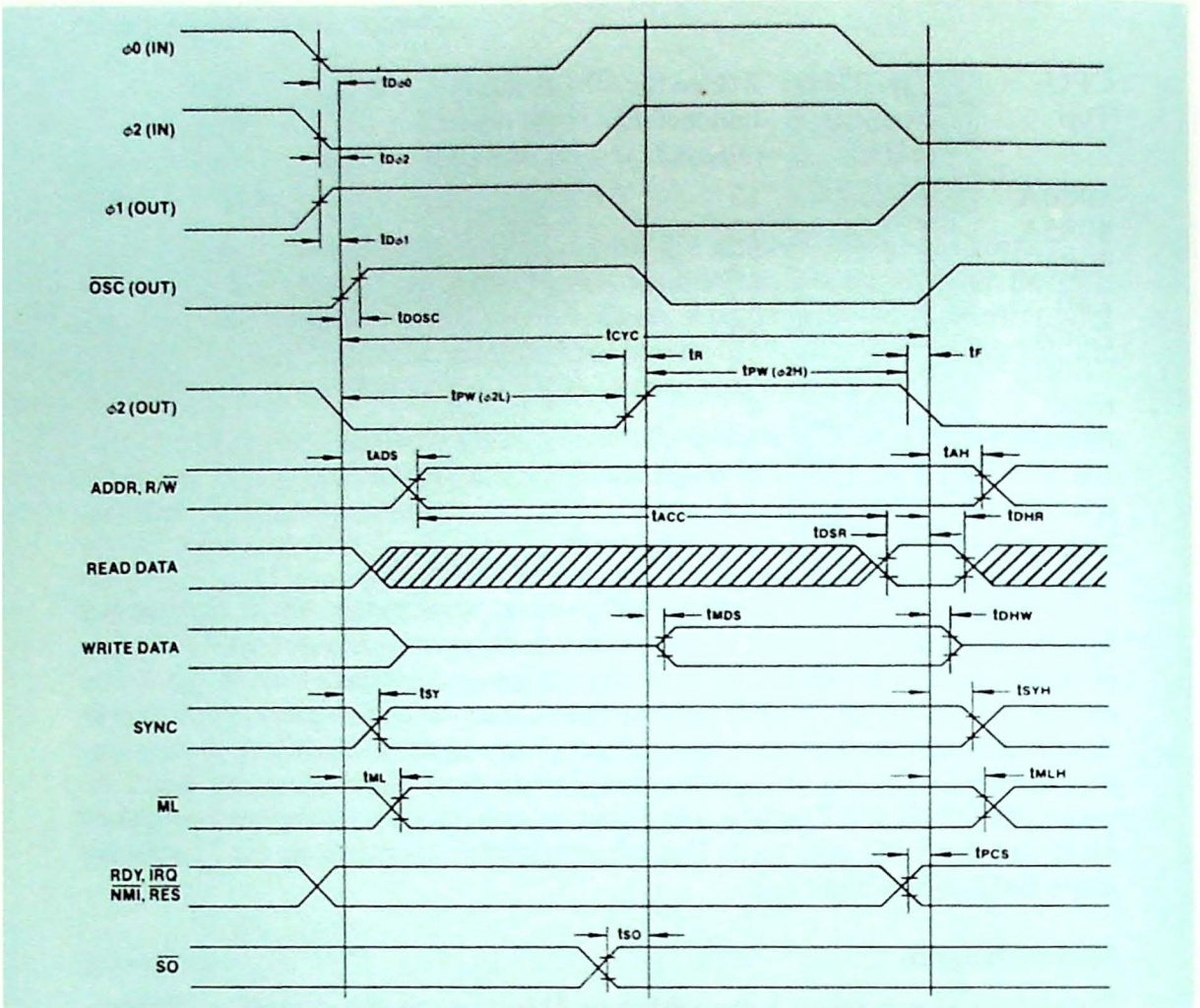
CPU-Typ	Typ. Takt-frequenz (MHz)	Zyklen für Ladebefehl (abs. Adr.)	Zeit (µs) für diesen Ladebefehl
8080A	2	13	6,5
8085A	3	13	4,33
8088	5	13	2,6
Z80	2	13	6,5
Z80A	4	13	3,25
Z80B	6	13	2,17
6800	1	4	4
6800B	2	4	2
6502	1	4	4
6502B	2	4	2
65C02	4	4	1

Da Lade- und Abspeicherungsbefehle einen wesentlichen Anteil an typischen Programmen haben, ist diese Tabelle schon ein erster Anhaltspunkt für die Geschwindigkeit von unterschiedlichen Prozessoren. Allerdings kann in der Praxis ein an das jeweilige Problem besonders gut angepaßter Befehlssatz ebenfalls zur Arbeitsgeschwindigkeit beitragen. Ebenso kann es erforderlich sein, durch sogenannte WAIT-Zyklen (künstliche Verzögerung durch Beschaltung des Wait-Eingangs der CPU) den Zugriff auf Speicher zu verlangsamen, deren Geschwindigkeit nicht der der CPU entspricht. Das erhöht natürlich zusätzlich die zur Ausführung eines Befehls benötigte Zeit.

Speicherzugriff

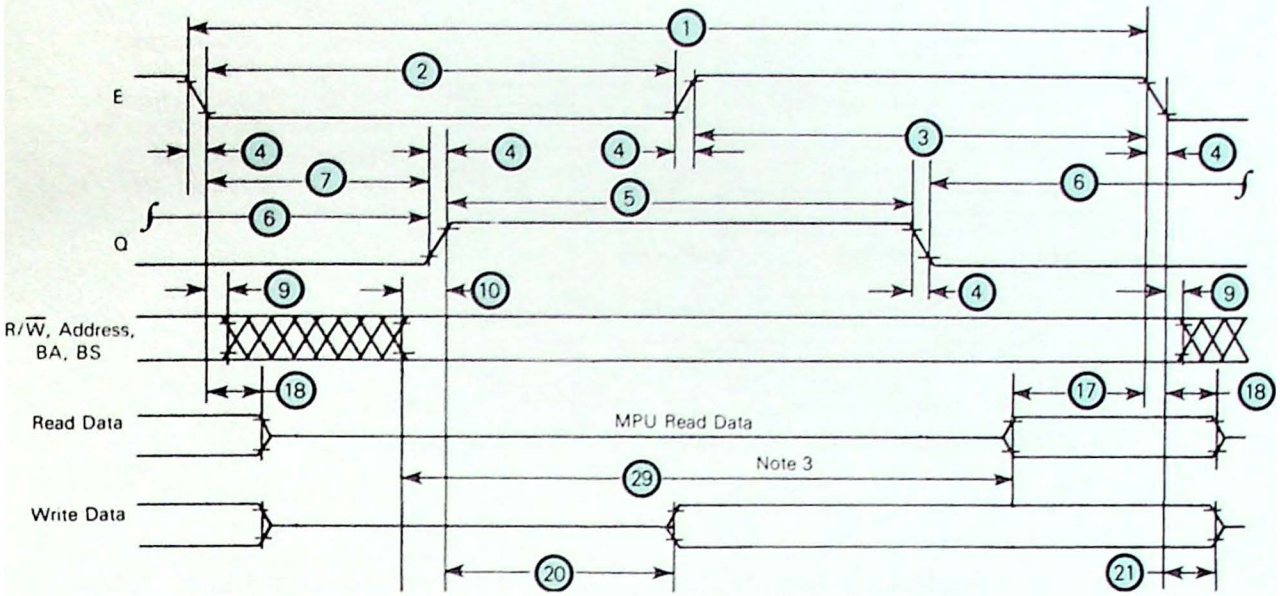
Damit wären wir schon beim zeitlichen Ablauf von Speicherzugriffen. Üblicherweise wird dieses „Timing" in Form von Impulsdiagrammen dargestellt *(Abb. 1.2.4.1 bis 1.2.4.5)*. Der prinzipielle Ablauf beim Lesen und Schreiben wurde ja schon in Kapitel 1.1.1 dargestellt. Zu beachten ist, daß die Wahl der Prozessor-Taktfrequenz auch einen direkten Einfluß auf die maximal zulässige Zugriffszeit der Speicher (RAM und ROM) bzw. der übrigen Systemkomponenten hat. Unter Zugriffszeit versteht man jene Zeit, die zwischen dem Anliegen der Adresse auf dem Adressenbus und der Antwort des Speichers mit Daten auf dem Datenbus vergeht. Die typischen Werte für einige CPUs sind:

CPU-Typ	Taktfrequenz (MHz)	max. Speicher-Zugriffszeit	CPU-Typ	Taktfrequenz (MHz)	max. Speicher-Zugriffszeit
8080A	2	500 ns	6800B	2	250 ns
8085A	3	500 ns	6502	1	575 ns
8088	5	500 ns	6502B	2	300 ns
Z80	2	750 ns	65C02	1	675 ns
Z80A	4	380 ns	65C02	2	310 ns
Z80B	6	250 ns	65C02	4	110 ns
6800	1	500 ns			



		1 MHz		2 MHz		3 MHz		4 MHz		
Parameter	Symbol	Min	Max	Min	Max	Min	Max	Min	Max	Unit
Delay Time, ϕ0 (IN) to ϕ2 (OUT)	$t_{D\phi0}$	—	100	—	100	—	100	—	100	nS
Delay Time, ϕ2 (IN) to ϕ2 (OUT)	$t_{D\phi2}$	—	75	—	75	—	75	—	75	nS
Delay Time, ϕ1 (OUT) to ϕ2 (OUT)	$t_{D\phi1}$	—	50	—	50	—	50	—	50	nS
Cycle Time	t_{CYC}	1.0	DC	0.50	DC	0.33	DC	0.25	DC	μS
Clock Pulse Width Low	$t_{PW(\phi2L)}$	470	—	240	—	160	—	115	—	nS
Clock Pulse Width High	$t_{PW(\phi2H)}$	470	—	240	—	160	—	115	—	nS
Fall Time, Rise Time	t_F, t_R	—	25	—	25	—	15	—	15	nS
Address Hold Time	t_{AH}	30	—	30	—	15	—	10	—	nS
Address Setup Time	t_{ADS}	225	—	140	—	110	—	90	—	nS
Access Time	t_{ACC}	650	—	310	—	170	—	110	—	nS
Read Data Hold Time	t_{DHR}	10	—	10	—	10	—	10	—	nS
Read Data Setup Time	t_{DSR}	100	—	50	—	50	—	50	—	nS
Write Data Delay Time	t_{MDS}	—	175	—	100	—	75	—	70	nS
Write Data Hold Time	t_{DHW}	30	—	30	—	30	—	30	—	nS
SO Setup Time	t_{SO}	100	—	50	—	35	—	25	—	nS
Processor Control Setup Time	t_{PCS}	200	—	200	—	150	—	120	—	nS

Abb. 1.2.4.1 Zeitdiagramm des CMOS-Prozessors 65SC02 von GTE Microcircuits



BUS TIMING CHARACTERISTICS (See Notes 1 and 2)

Ident. Number	Characteristics	Symbol	MC6809		MC68A09		MC68B09		Unit
			Min	Max	Min	Max	Min	Max	
1	Cycle Time (See Note 5)	t_{cyc}	1.0	10	0.667	10	0.5	10	µs
2	Pulse Width, E Low	PW_{EL}	430	5000	280	5000	210	5000	ns
3	Pulse Width, E High	PW_{EH}	450	15500	280	15700	220	15700	ns
4	Clock Rise and Fall Time	t_r, t_f	–	25	–	25	–	20	ns
5	Pulse Width, Q High	PW_{QH}	430	5000	280	5000	210	5000	ns
6	Pulse Width, Q Low	PW_{QL}	450	15500	280	15700	220	15700	ns
7	Delay Time, E to Q Rise	t_{AVS}	200	250	130	165	80	125	ns
9	Address Hold Time* (See Note 4)	t_{AH}	20	–	20	–	20	–	ns
10	BA, BS, R/W, and Address Valid Time to Q Rise	t_{AQ}	50	–	25	–	15	–	ns
17	Read Data Setup Time	t_{DSR}	80	–	60	–	40	–	ns
18	Read Data Hold Time*	t_{DHR}	10	–	10	–	10	–	ns
20	Data Delay Time from Q	t_{DDQ}	–	200	–	140	–	110	ns
21	Write Data Hold Time*	t_{DHW}	30	–	30	–	30	–	ns
29	Usable Access Time (See Note 3)	t_{ACC}	695	–	440	–	330	–	ns
	Processor Control Setup Time (MRDY, Interrupts, $\overline{DMA/BREQ}$, $\overline{HALT}$, $\overline{RESET}$) (Figures 6, 8, 9, 10, 12, and 13)	t_{PCS}	200	–	140	–	110	–	ns
	Crystal Oscillator Start Time (Figures 6 and 7)	t_{RC}	–	100	–	100	–	100	ms
	Processor Control Rise and Fall Time (Figures 6 and 8)	t_{PCr}, t_{PCf}	–	100	–	100	–	100	ns

* Address and data hold times are periodically tested rather than 100% tested.

NOTES:
1. Voltage levels shown are $V_L \leq 0.4$ V, $V_H \geq 2.4$ V, unless otherwise specified.
2. Measurement points shown are 0.8 V and 2.0 V, unless otherwise specified.
3. Usable access time is computed by: 1 – 4 – 7 max + 10 – 17.
4. Hold time ((9)) for BA and BS is not specified.
5. Maximum t_{cyc} during MRDY or $\overline{DMA/BREQ}$ is 16 µs.

Abb. 1.2.4.2 Zeitdiagramm des Prozessors 6809 von Motorola

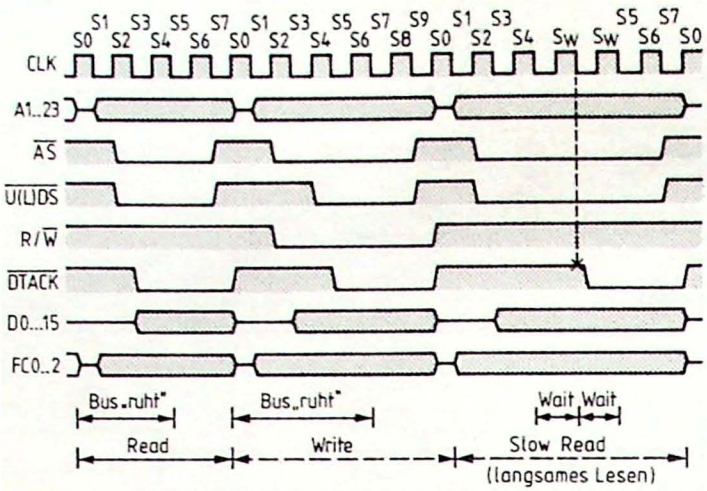


Abb. 1.2.4.3
Je nach Geschwindigkeit der angeschlossenen Speicher-Bausteine kann der Zugriff beim 68000 dank des asynchronen Busverhaltens schnell oder langsam erfolgen

Vergleicht man dies mit dem Zeitbedarf für die Abarbeitung eines Ladebefehls bei einer bestimmten vorgegebenen Speicher-Zugriffszeit von z.B. 300 ns, so darf z.B. der Z80A mit 4 MHz und der 6502 mit 2 MHz eingesetzt werden, woraus sich ein Zeitbedarf von 3,25 µs beim Z80 und 2 µs beim 6502 ergibt. (Daraus läßt sich auch schließen, daß ein Z80B bei 6 MHz Taktfrequenz etwa so schnell arbeitet wie ein 6502 bei 2 MHz.) Man beachte, daß es beim 6502, besonders bei der CMOS-Version 65C02, gelungen ist, durch entsprechende Auslegung der internen Logik eine verhältnismäßig hohe Zugriffszeit bei gleicher Taktfrequenz oder, umgekehrt, eine hohe Taktfrequenz bei gleicher Zugriffszeit zuzulassen.

Der Zugriff auf Ein/Ausgabe-Bausteine verläuft prinzipiell ähnlich wie ein Speicherzugriff: Wenn Daten ausgegeben werden sollen, erfolgt dies mit einem Schreibvorgang in ein I/O-Register; bei Daten-Eingaben ist dagegen ein Lesevorgang nötig. Während CPUs wie 8080, 8085, 8088, Z80 und Z8000 für Ein/Ausgabe-Operationen sowohl besondere Befehle (IN/OUT) als auch spezielle Steuerleitungen (IORQ, ST n) besitzen, sprechen andere wie 6800, 6809, 6502 und 68000 I/O-Bausteine exakt wie normale Speicher an.

Das führt dazu, daß z.B. beim Z80 der volle adressierbare Bereich von 64 KByte mit Speicher ausgefüllt werden kann, während z.B. beim 6502 ein Teil dieses Bereichs für Ein-/Ausgabebausteine reserviert werden muß.

Eine spezielle Eigenschaft von Z80 und Z8000 ist, daß diese Prozessoren über eine eingebaute Logik zum automatischen regelmäßigen Auffrischen dynamischer RAMs enthalten (Refresh; siehe Kapitel über Schreib-Lese-Speicher). Sie besteht im wesentlichen aus einem Refresh-Zähler, der zyklisch alle Speicher-Adressengruppen anspricht, während die CPU gerade mit internen Dingen beschäftigt ist. Bei anderen CPUs ist dafür eine externe Schaltung nötig.

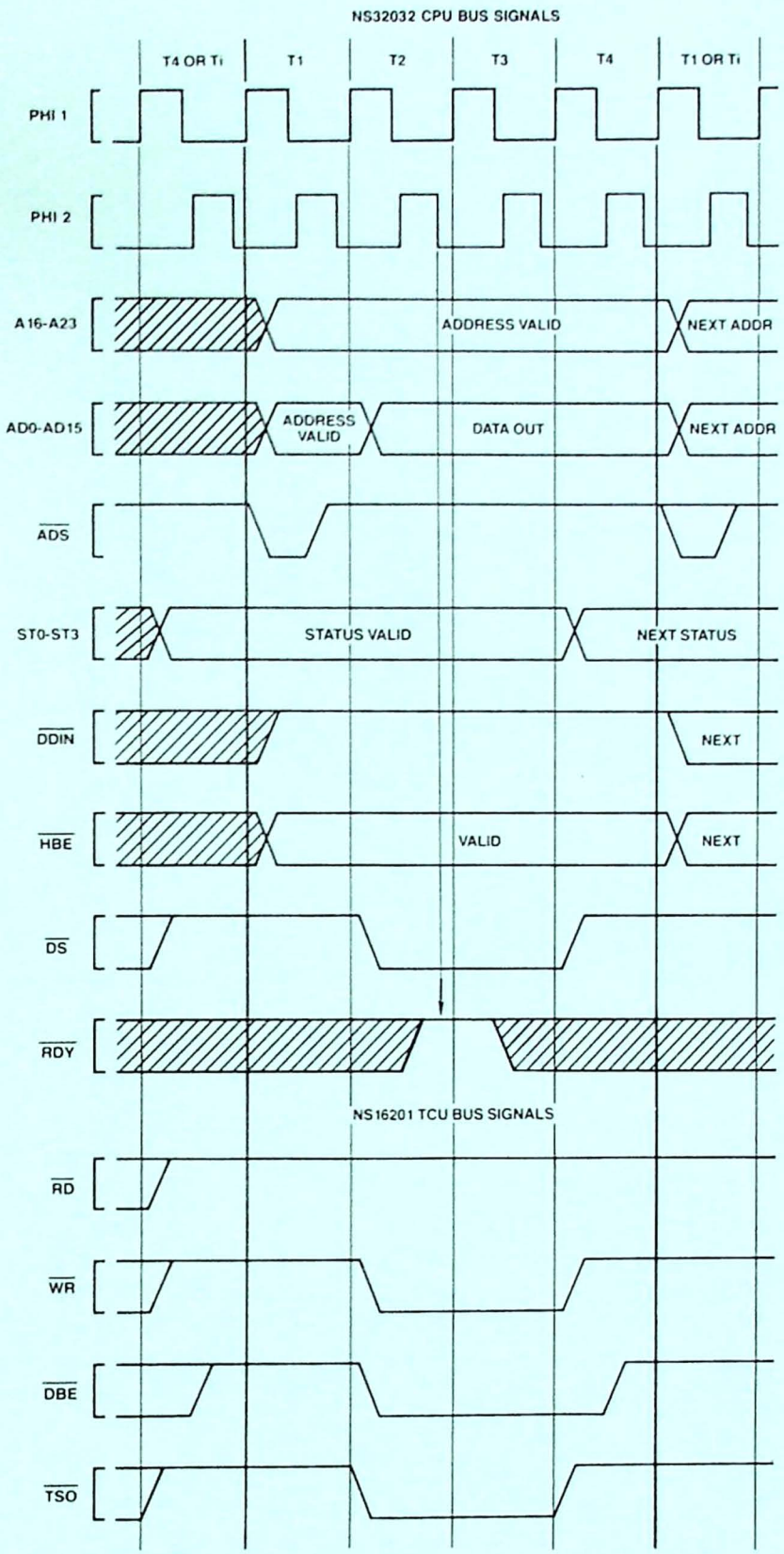
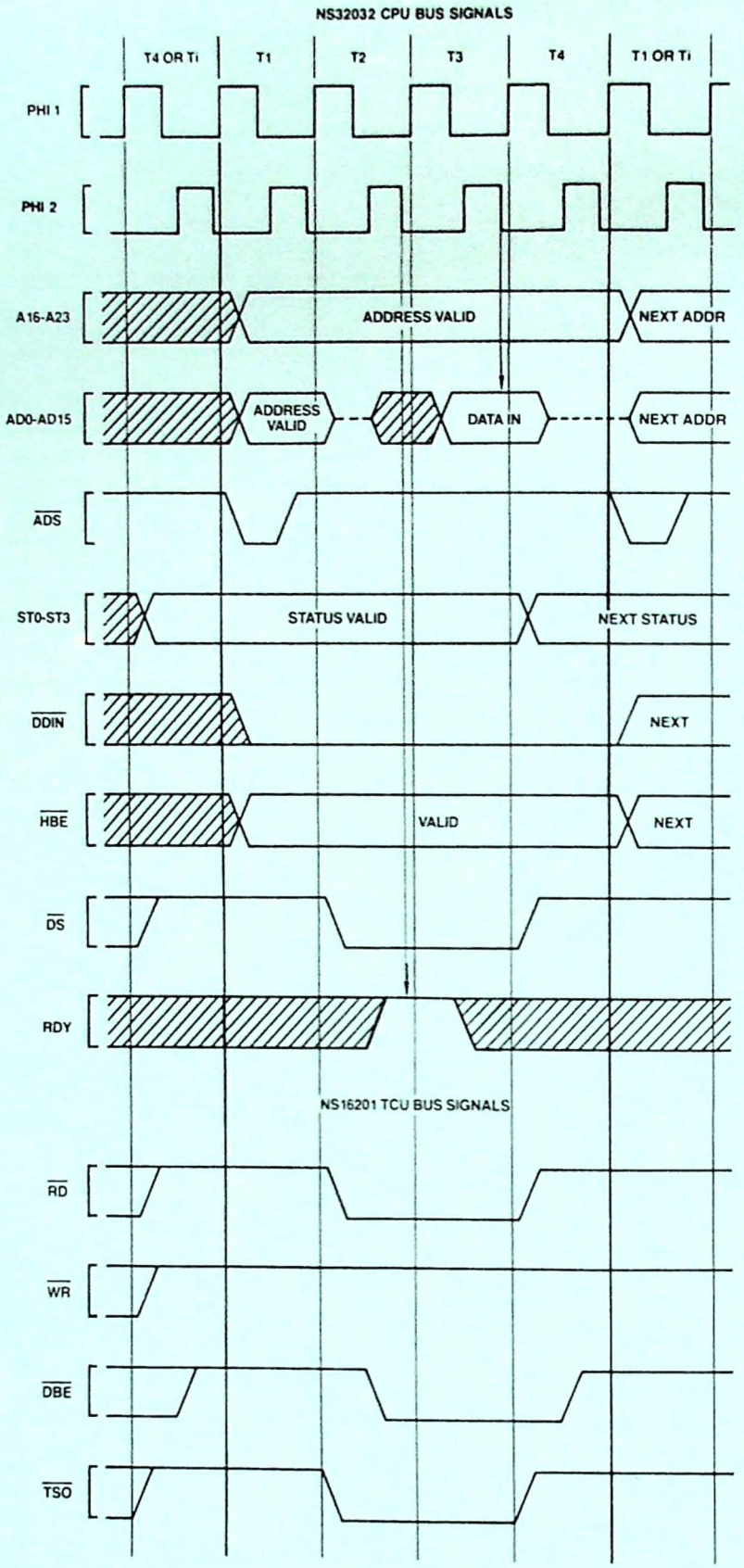


Abb. 1.2.4.4
Schreib- und Lese-zyklen des Prozessors 32032 (Serie 32)


NS32032 CPU BUS SIGNALS
T4 OR Ti
T1
T2
T3
T4
T1 OR Ti
PHI 1
PHI 2
A16-A23
ADDRESS VALID
NEXT ADDR
AD0-AD15
ADDRESS VALID
DATA IN
NEXT ADDR
ADS
ST0-ST3
STATUS VALID
NEXT STATUS
DDIN
NEXT
HBE
VALID
NEXT
DS
RDY
NS16201 TCU BUS SIGNALS
RD
WR
DBE
TSO

zu Abb. 1.2.4.4

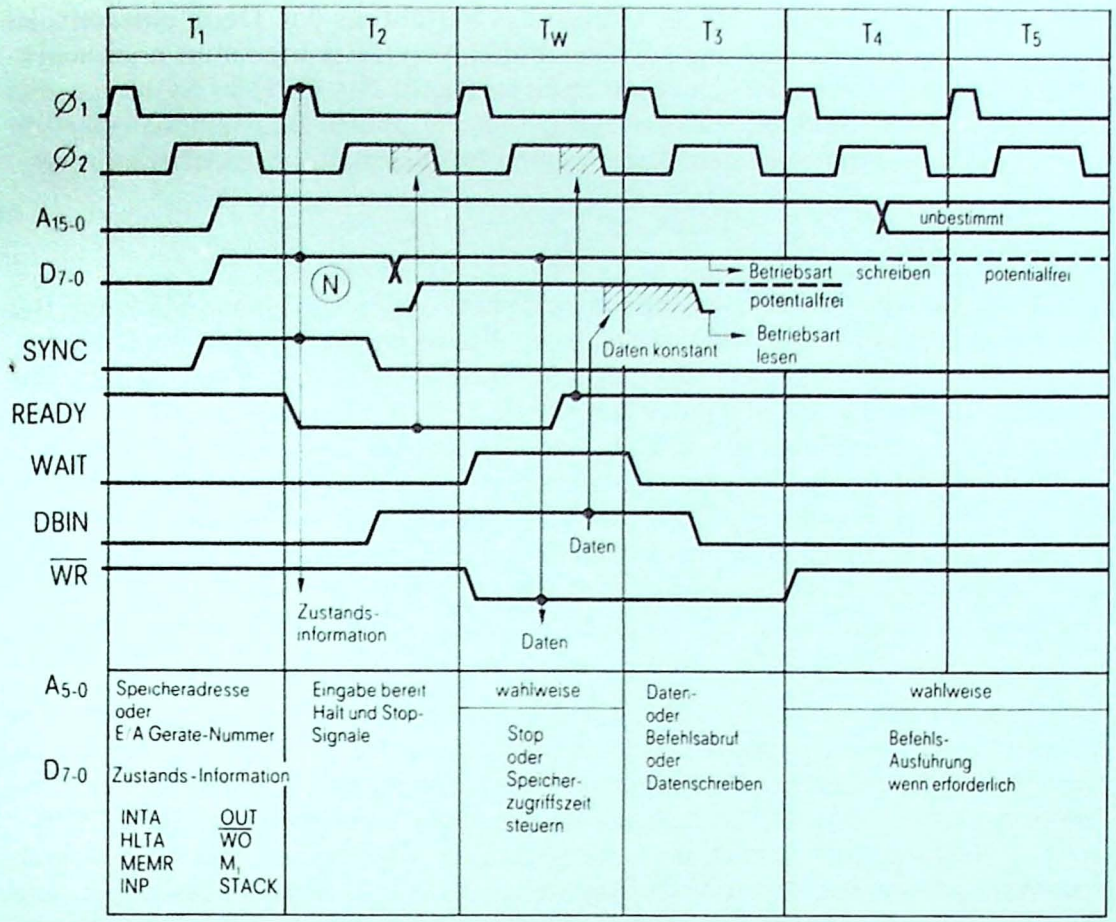


Abb. 1.2.4.5 Zeitdiagramm des Prozessors 8080 von Intel

Pipelining

Schließlich sei noch auf eine trickreiche Einrichtung in manchen CPUs hingewiesen, die ebenfalls Zeit sparen kann. Eingeführt wurde sie erstmals beim 6502: Noch während der Prozessor mit der Abarbeitung eines Befehls beschäftigt ist, legt er die auf ihn folgende Speicheradresse auf den Adressenbus und liest das dort stehende Byte in einen internen Puffer. Dies nennt man „Pipelining"; es hat den Vorteil, daß dadurch effektiv an Speicherzugriffszeit gespart wird. Denn die CPU kennt das nächste Byte ja schon, bevor sie wieder auf den Speicher zugreift. Nur in einem Fall versagt die Methode natürlich — nämlich bei einem Sprungbefehl, wenn das Programm also nicht bei der nächsten Speicherstelle, sondern an einer anderen Adresse fortgeführt werden soll.

Auch die CPUs 8088 und 8086 kennen dieses Pipelining; der Hersteller Intel nennt es allerdings „Instruction Queue" (Befehlsschlange). Während beim 6502 der Puffer nur 1 Byte umfaßt, sind es beim 8088 vier und beim 8086 sogar sechs.

Dies stellt aber auch die natürliche Grenze des Verfahrens dar. Denn einerseits ist die Geschwindigkeits-Erhöhung bei einer Puffer-Vergrößerung kaum noch merklich. Andererseits wirkt der dauernde Speicherzugriff der CPU in Systemen, bei denen mit DMA (direkter Speicherzugriff von Peripherie-Bausteinen) oder mit mehreren Prozessoren auf einem gemeinsamen Bus gearbeitet wird, eher störend.

Cache-Speicher

Eine fortentwickelte Version des Pipelining ist der sogenannte Cache-Speicher. Bei ihm bleibt eine gewisse Anzahl auch schon abgearbeiteter Befehle noch im Befehls-Puffer der CPU stehen. Das hat den Vorteil, daß bei kleinen Programmschleifen alle Befehle innerhalb der Schleife noch im Puffer stehen und somit zur mehrmaligen Abarbeitung des Schleifenstücks keinerlei Zugriff auf den externen Programmspeicher nötig ist. Das wiederum kann die effektive Geschwindigkeit einer CPU erheblich erhöhen, da die Schleifen oft die meiste Zeit innerhalb eines Programms beanspruchen. Je umfangreicher der Cache-Speicher ist, desto länger darf das Schleifenstück sein, ohne einen externen Speicherzugriff zu benötigen. Der 8-Bit-Prozessor Z800, eine Fortentwicklung des Z80, und die 32-Bit-CPU 68020 sind zwei Beispiele für Mikroprozessoren mit Cache-Speicher.

Benötigt beispielsweise die CPU Z800 ein Byte, so prüft sie zunächst, ob es schon im Cache-Speicher steht, und leitet nur dann, wenn das nicht der Fall ist, einen externen Speicherzugriff ein. Die am längsten nicht mehr angesprochene 16-Bit-Gruppe wird gelöscht. Das nun gelesene Byte wird nicht nur von der CPU benutzt, sondern zusätzlich auch im Cache-Speicher abgelegt, so daß es für einen späteren weiteren Zugriff zur Verfügung steht. Da es sehr wahrscheinlich ist, daß die CPU nach einem bestimmten Byte auch das folgende benötigt, liest der Prozessor, während er gerade einen Befehl abarbeitet, sicherheitshalber so viele der folgenden Bytes in den Cache-Speicher, wie er schafft (natürlich nicht mehr, als in den Cache-Speicher passen, beim Z800 maximal 16 Bytes). Die effektive Geschwindigkeit einer CPU kann sich durch den Cache-Speicher je nach Software um den Faktor 2...8 erhöhen.

1.2.5 Befehlssätze einiger Mikroprozessoren

Da Mikroprozessoren in binärer Logik aufgebaut sind, können sie zunächst einmal nur Dualzahlen als Befehle verstehen. So bedeutet z.B. der Befehl 10101001 beim 6502, daß der Akku mit dem Wert geladen werden soll, der in der auf diesen „Operationscode" folgenden Speicherzelle steht.

Normalerweise schreibt man die Operationscodes aber nicht als Dual-, sondern als Hexadezimalzahl, hier also A9. Auch das ist aber weder sehr aussagekräftig noch läßt es sich gut merken.

Deshalb führte man die sogenannten Mnemonics ein. Das sind Abkürzungen aus wenigen Buchstaben, die in englischer Sprache das beschreiben, was der Befehl tatsächlich tut. In obigem Beispiel heißt das Mnemonic LDA; das ist die Abkürzung von Load Accumulator, also „Lade Akku".

Natürlich ist der Prozessor nicht in der Lage, solche Mnemonics direkt zu verstehen. Schreibt man ein Programm daher in Form von Mnemonics, so muß man es anschließend in hexadezimale Operationscodes übersetzen. Das kann entweder (ziemlich mühsam) von Hand geschehen oder automatisiert mit einem besonderen Übersetzungsprogramm, dem Assembler. Auf ihn werden wir in einem späteren Kapitel noch ausführlicher zu sprechen kommen.

Die Zuordnung von Operationscodes zu Mnemonics muß nicht eindeutig sein. Vielmehr ist es möglich, daß es zu einem Mnemonic mehrere Operationscodes gibt, je nachdem, auf welchen Operand sich der Befehl bezieht (Daten, Adresse, Registerinhalt usw.). Diese Unterscheidung nennt man „Adressierungsarten". Neben der Zahl der vorhandenen mnemonischen Befehle spielt daher auch die Zahl der möglichen Adressierungsarten eine wesentliche Rolle bei der Beurteilung der Leistungsfähigkeit von Mikroprozessoren. Der Trend geht heute sogar dahin, mit möglichst wenig Mnemonics auszukommen, um eine leichte Erlernbarkeit des Befehlssatzes zu gewährleisten, andererseits aber für jeden Befehl jede sinnvolle Adressierungsart zur Verfügung zu stellen. So aufgebaute Befehlssätze nennt man „orthogonal".

Die folgende Übersicht zeigt die typischen Adressierungsarten des Prozessors 65C02:

Adressierung	*Mnem.Schreibw.*	*Wirkung*
Immediate	LDA #5	Der Akku wird mit der Zahl 5 geladen.
Absolut	LDA 1398	Der Akku wird mit dem Inhalt der Zelle 1398 geladen.
Zero Page	LDA 56	Wie Absolut, aber mit nur einem Byte als Adresse.
Indirekt	LDA (56)	Der Akku wird mit dem Inhalt der Zelle geladen, deren Adresse in den Zellen 56 und 57 steht.
Implizit	TXA	Transfer X-Register in Akku; der Befehl beinhaltet die Adressierungsart selbst.
Indiziert	LDA 0200,X	Der Akku wird mit dem Inhalt der Zelle geladen, deren Adresse sich aus 0200 plus dem Inhalt des X-Registers ergibt.
Nachindiziert	LDA (56),Y	Zu der in den Zellen 56 und 57 stehenden Adresse wird der Inhalt des Y-Registers addiert. Das Ergebnis ist die Adresse, mit der der Akku geladen wird.

Vorindiziert	LDA (56,X)	Zu dem Wert 56 wird der Inhalt des X-Registers (z.B. 02) addiert, das Ergebnis ist hier 58. Nun wird der Akku mit dem Inhalt jener Zelle geladen, die in den Zellen 58 und 59 steht.
Relativ	BRA XX	Branch-Befehl: Sprung um XX+2 Bytes nach vorwärts, wenn XX kleiner als hex 80 ist, und um hex FE-XX Bytes nach rückwärts, wenn XX = 80...FF ist.
Zero-Page-indiziert	LDA 56,X	Wie indiziert, jedoch mit nur einem Byte als Adresse.
Einzel-Bit	SMB 1234,5	In der Speicherzelle 1234 wird das Bit Nr. 5 auf 1 gesetzt.
Indirekt,X	JMP (1234),X	Es erfolgt ein Sprung zu jener Adresse, die sich aus der Addition von 1234 und dem Inhalt des X-Registers ergibt.

Der älteste 8-Bit-Prozessor, Intels 8008, kannte nur eine einzige Möglichkeit, auf Daten im Speicher zuzugreifen: Bei ihm war es nötig, zuerst die jeweilige Adresse ins HL-Register zu laden. Der Ladebefehl selbst (implizite Adressierung) verwendete dann den Inhalt von HL als Adresse. Insgesamt verfügte der 8008 also nur über die Adressierungsarten Impliziert (implied) und Immediate.

Beim 8080 bzw. 8085 kam dann noch die absolute (von Intel „direkt" genannte) Adressierungsart hinzu. Die Entwickler des Z80 von Zilog verwendeten schließlich die IX/IY-Register für indizierte Adressierung und sahen auch relative Sprungbefehle vor.

Die segmentierte Adressierung des 8088 und 8086 wurde schon im Kapitel über Register beschrieben. Sie ist eine notwendige Krücke, weil bei diesen Prozessoren einerseits der Programmzähler nur 16 Bit lang ist, andererseits aber ein größerer Speicherbereich als 64 KByte ansprechbar sein sollte. Mit den Registern SI oder DI ist eine indizierte Adressierung mit 8- oder 16-Bit-Basisadresse möglich (je nach Operationscode). Außerdem gibt es hier noch die Adressierung relativ zu einer Basisadresse (auch indiziert mit BX), die in den Registern DS oder SS stehen kann. Wie die 6502- verwendet die 8086-Familie ausschließlich die relative Adressierung bei bedingten Sprungbefehlen (Branch), ebenso ist eine indirekte Adressierung möglich.

Beim Z8000 existieren, abgesehen von der Segmentierung, acht Adressierungs-Möglichkeiten:

1. Register-Adressierung: Der Operand steht in einem Register.
2. Immediate: Der Operand folgt im Programm auf den Operationscode.
3. Implizit (von Zilog indirekt genannt): Die Adresse des Operanden steht in einem Register.
4. Absolut (von Zilog „direkt" genannt): Die Adresse des Operanden steht im Befehl.
5. Indiziert: Die Basisadresse des Operanden steht im Befehl, die Distanzadresse in einem Register.
6. Relativ: Die Basisadresse steht im Programmzähler, die Distanzadresse im Befehl.
7. Basis-absolut: Die Basisadresse des Operanden steht in einem Register, die Distanzadresse im Befehl.
8. Basis-indiziert: Sowohl Basis- als auch Distanzadresse stehen in jeweils einem Register.

Die meisten CPU-Typen enthalten eine Untermenge oder Kombination der bisher genannten Adressierungsarten. Anders bei den Befehlen: Hier gibt es ganz erhebliche Unterschiede in der Syntax, also in der Schreibweise. Grob lassen sich die Befehle aber immer in folgende Gruppen einteilen:

1. Lade- und Speicherbefehle (LD = Load, ST = Store, MOV = Move, EX = Exchange, PUSH, POP, PHA, PLA, TAX, IN, OUT usw.)
2. Arithmetische und logische Befehle (OR, AND, BIT, ADD, ADC, SUB, SBC, MUL, INC, DEC, INX, INY usw.)
3. Bit-Manipulation (ROL, ROR, LSR, ASL, RRCA, SLA, SRA, TSB usw.)
4. Sprungbefehle (JMP, BNE, JSR, RTS, BRK, JP, JR, CALL, RET usw.)
5. CPU-Kontrollbefehle (SEI, CLI, SED, CLD, DI, EI, HALT, IM usw.)

Die *Abb. 1.2.5.1 bis 1.2.5.11* dieses Kapitels zeigen die Befehlssätze einiger heute verwendeten CPU-Typen. Wie besonders das Beispiel 8086 zeigt, gibt es besonders bei 16-Bit-Prozessoren derart viele Operationscodes, daß dieses Buch keinen Anspruch auf Vollständigkeit erheben kann. Statt dessen sei auf die Hersteller-Adressen in Kapitel 6.2 verwiesen.

00	–	40	HALT	80	ADDZ, 0	C0	NOP
01	LODZ, 1	41	ANDZ, 1	81	ADDZ, 1	C1	STRZ, 1
02	LODZ, 2	42	ANDZ, 2	82	ADDZ, 2	C2	STRZ, 2
03	LODZ, 3	43	ANDZ, 3	83	ADDZ, 3	C3	STRZ, 3
04	LODI, 0	44	ANDI, 0	84	ADDI, 0	C4	–
05	LODI, 1	45	ANDI, 1	85	ADDI, 1	C5	–
06	LODI, 2	46	ANDI, 2	86	ADDI, 2	C6	–
07	LODI, 3	47	ANDI, 3	87	ADDI, 3	C7	–
08	LODR, 0	48	ANDR, 0	88	ADDR, 0	C8	STRR, 0
09	LODR, 1	49	ANDR, 1	89	ADDR, 1	C9	STRR, 1
0A	LODR, 2	4A	ANDR, 2	8A	ADDR, 2	CA	STRR, 2
0B	LODR, 3	4B	ANDR, 3	8B	ADDR, 3	CB	STRR, 3
0C	LODA, 0	4C	ANDA, 0	8C	ADDA, 0	CC	STRA, 0
0D	LODA, 1	4D	ANDA, 1	8D	ADDA, 1	CD	STRA, 1
0E	LODA, 2	4E	ANDA, 2	8E	ADDA, 2	CE	STRA, 2
0F	LODA, 3	4F	ANDA, 3	8F	ADDA, 3	CF	STRA, 3
10	–	50	RRR, 0	90	–	D0	RRL, 0
11	–	51	RRR, 1	91	–	D1	RRL, 1
12	SPSU	52	RRR, 2	92	LPSU	D2	RRL, 2
13	SPSL	53	RRR, 3	93	LPSL	D3	RRL, 3
14	RETC, *0*	54	REDE, 0	94	DAR, 0	D4	WRTE, 0
15	RETC, *1*	55	REDE, 1	95	DAR, 1	D5	WRTE, 1
16	RETC, *2*	56	REDE, 2	96	DAR, 2	D6	WRTE, 2
17	RETC, *3*	57	REDE, 3	97	DAR, 3	D7	WRTE, 3
18	BCTR, *0*	58	BRNR, 0	98	BCFR, *0*	D8	BIRR, 0
19	BCTR, *1*	59	BRNR, 1	99	BCFR, *1*	D9	BIRR, 1
1A	BCTR, *2*	5A	BRNR, 2	9A	BCFR, *2*	DA	BIRR, 2
1B	BCTR, *3*	5B	BRNR, 3	9B	ZBRR	DB	BIRR, 3
1C	BCTA, *0*	5C	BRNA, 0	9C	BCFA, *0*	DC	BIRA, 0
1D	BCTA, *1*	5D	BRNA, 1	9D	BCFA, *1*	DD	BIRA, 1
1E	BCTA, *2*	5E	BRNA, 2	9E	BCFA, *2*	DE	BIRA, 2
1F	BCTA, *3*	5F	BRNA, 3	9F	BXA	DF	BIRA, 3
20	EORZ, 0	60	IORZ, 0	A0	SUBZ, 0	E0	COMZ, 0
21	EORZ, 1	61	IORZ, 1	A1	SUBZ, 1	E1	COMZ, 1
22	EORZ, 2	62	IORZ, 2	A2	SUBZ, 2	E2	COMZ, 2
23	EORZ, 3	63	IORZ, 3	A3	SUBZ, 3	E3	COMZ, 3
24	EORI, 0	64	IORI, 0	A4	SUBI, 0	E4	COMI, 0
25	EORI, 1	65	IORI, 1	A5	SUBI, 1	E5	COMI, 1
26	EORI, 2	66	IORI, 2	A6	SUBI, 2	E6	COMI, 2
27	EORI, 3	67	IORI, 3	A7	SUBI, 3	E7	COMI, 3
28	EORR, 0	68	IORR, 0	A8	SUBR, 0	E8	COMR, 0
29	EORR, 1	69	IORR, 1	A9	SUBR, 1	E9	COMR, 1
2A	EORR, 2	6A	IORR, 2	AA	SUBR, 2	EA	COMR, 2
2B	EORR, 3	6B	IORR, 3	AB	SUBR, 3	EB	COMR, 3
2C	EORA, 0	6C	IORA, 0	AC	SUBA, 0	EC	COMA, 0
2D	EORA, 1	6D	IORA, 1	AD	SUBA, 1	ED	COMA, 1
2E	EORA, 2	6E	IORA, 2	AE	SUBA, 2	EE	COMA, 2
2F	EORA, 3	6F	IORA, 3	AF	SUBA, 3	EF	COMA, 3
30	REDC, 0	70	REDD, 0	B0	WRTC, 0	F0	WRTD, 0
31	REDC, 1	71	REDD, 1	B1	WRTC, 1	F1	WRTD, 1
32	REDC, 2	72	REDD, 2	B2	WRTC, 2	F2	WRTD, 2
33	REDC, 3	73	REDD, 3	B3	WRTC, 3	F3	WRTD, 3
34	RETE, *0*	74	CPSU	B4	TPSU	F4	TMI, 0
35	RETE, *1*	75	CPSL	B5	TPSL	F5	TMI, 1
36	RETE, *2*	76	PPSU	B6	–	F6	TMI, 2
37	RETE, *3*	77	PPSL	B7	–	F7	TMI, 3
38	BSTR, *0*	78	BSNR, 0	B8	BSFR, *0*	F8	BDRR, 0
39	BSTR, *1*	79	BSNR, 1	B9	BSFR, *1*	F9	BDRR, 1
3A	BSTR, *2*	7A	BSNR, 2	BA	BSFR, *2*	FA	BDRR, 2
3B	BSTR, *3*	7B	BSNR, 3	BB	ZBSR	FB	BDRR, 3
3C	BSTA, *0*	7C	BSNA, 0	BC	BSFA, *0*	FC	BDRA, 0
3D	BSTA, *1*	7D	BSNA, 1	BC	BSFA, *1*	FD	BDRA, 1
3E	BSTA, *2*	7E	BSNA, 2	BE	BSFA, *2*	FE	BDRA, 2
3F	BSTA, *3*	7F	BSNA, 3	BF	BSXA	FF	BDRA, 3

Kursiv geschriebene Ziffern hinter dem Kürzel bedeuten, daß der Bedingungscode die Ausführung des Befehls bestimmt; sonst bezeichnet die Ziffer das beteiligte Register

Abb. 1.2.5.1 Befehle des schon etwas betagten Prozessors 2650 von Valvo/Signetics

00	NOP		2B	DCX	H	56	MOV	D,M	81	ADD	C	AC	XRA	H	D7	RST	2
01	LXI	B,(16)	2C	INR	L	57	MOV	D,A	82	ADD	D	AD	XRA	L	D8	RC	
02	STAX	B	2D	DCR	L	58	MOV	E,B	83	ADD	E	AE	XRA	M	D9	...	
03	INX	B	2E	MVI	L,(8)	59	MOV	E,C	84	ADD	H	AF	XRA	A	DA	JC	Adr.
04	INR	B	2F	CMA		5A	MOV	E,D	85	ADD	L	B0	ORA	B	DB	IN	(8)
05	DCR	B	30			5B	MOV	E,E	86	ADD	M	B1	ORA	C	DC	CC	Adr.
06	MVI	B,(8)	31	LXI	SP,(16)	5C	MOV	E,H	87	ADD	A	B2	ORA	D	DD	...	
07	RLC		32	STA	Adr.	5D	MOV	E,L	88	ADC	B	B3	ORA	E	DE	SBI	D8
08			33	INX	SP	5E	MOV	E,M	89	ADC	C	B4	ORA	H	DF	RST	3
09	DAD	B	34	INR	M	5F	MOV	E,A	8A	ADC	D	B5	ORA	L	E0	RPO	
0A	LDAX	B	35	DCR	M	60	MOV	H,B	8B	ADC	E	B6	ORA	M	E1	POP	H
0B	DCX	B	36	MVI	M,(8)	61	MOV	H,C	8C	ADC	H	B7	ORA	A	E2	JPO	Adr.
0C	INR	C	37	STC		62	MOV	H,D	8D	ADC	L	B8	CMP	B	E3	XTHL	
0D	DCR	C	38			63	MOV	H,E	8E	ADC	M	B9	CMP	C	E4	CPO	Adr.
0E	MVI	C,(8)	39	DAD	SP	64	MOV	H,H	8F	ADC	A	BA	CMP	D	E5	PUSH	H
0F	RRC		3A	LDA	Adr.	65	MOV	H,L	90	SUB	B	BB	CMP	E	E6	ANI	(8)
10			3B	DCX	SP	66	MOV	H,M	91	SUB	C	BC	CMP	H	E7	RST	4
11	LXI	D,(16)	3C	INR	A	67	MOV	H,A	92	SUB	D	BD	CMP	L	E8	RPE	
12	STAX	D	3D	DCR	A	68	MOV	L,B	93	SUB	E	BE	CMP	M	E9	PCHL	
13	INX	D	3E	MVI	A,(8)	69	MOV	L,C	94	SUB	H	BF	CMP	A	EA	JPE	Adr.
14	INR	D	3F	CMC		6A	MOV	L,D	95	SUB	L	C0	RNZ		EB	XCHG	
15	DCR	D	40	MOV	B,B	6B	MOV	L,E	96	SUB	M	C1	POP	B	EC	CPE	Adr.
16	MVI	D,(8)	41	MOV	B,C	6C	MOV	L,H	97	SUB	A	C2	JNZ	Adr.	ED	...	
17	RAL		42	MOV	B,D	6D	MOV	L,L	98	SBB	B	C3	JMP	Adr.	EE	XRI	(8)
18			43	MOV	B,E	6E	MOV	L,M	99	SBB	C	C4	CNZ	Adr.	EF	RST	5
19	DAD	D	44	MOV	B,H	6F	MOV	L,A	9A	SBB	D	C5	PUSH	B	F0	RP	
1A	LDAX	D	45	MOV	B,L	70	MOV	M,B	9B	SBB	E	C6	ADI	(8)	F1	POP	PSW
1B	DCX	D	46	MOV	B,M	71	MOV	M,C	9C	SBB	H	C7	RST	0	F2	JF	Adr.
1C	INR	E	47	MOV	B,A	72	MOV	M,D	9D	SBB	L	C8	RZ		F3	DI	
1D	DCR	E	48	MOV	C,B	73	MOV	M,E	8E	SBB	M	C9	RET	Adr.	F4	CP	Adr.
1E	MVI	E,(8)	49	MOV	C,C	74	MOV	M,H	9F	SBB	A	CA	JZ		F5	PUSH	PSW
1F	RAR		4A	MOV	D,D	75	MOV	M,L	A0	ANA	B	CB			F6	ORI	(8)
20			4B	MOV	C,E	76	HLT		A1	ANA	C	CC	CZ	Adr.	F7	RST	6
21	LXI	H,(16)	4C	MOV	C,H	77	MOV	M,A	A2	ANA	D	CD	CALL	Adr.	F8	RM	
22	SHLD	Adr.	4D	MOV	C,L	78	MOV	A,B	A3	ANA	E	CE	ACI	(8)	F9	SPHL	
23	INX	H	4E	MOV	C,M	79	MOV	A,C	A4	ANA	H	CF	RST	1	FA	JM	Adr.
24	INR	H	4F	MOV	C,A	7A	MOV	A,D	A5	ANA	L	D0	RNC		FB	EI	
25	DCR	H	50	MOV	D,B	7B	MOV	A,E	A6	ANA	M	D1	POP	D	FC	CM	Adr.
26	MVI	H,(08)	51	MOV	D,C	7C	MOV	A,H	A7	ANA	A	D2	JNC	Adr.	FD		
27	DAA		52	MOV	D,D	7D	MOV	A,L	A8	XRA	B	D3	OUT	(8)	FE	CPI	D8
28			53	MOV	D,E	7E	MOV	A,M	A9	XRA	C	D4	CNC	Adr.	FF	RST	7
29	DAD	H	54	MOV	D,H	7F	MOV	A,A	AA	XRA	D	D5	PUSH	D			
2A	LHLD	Adr.	55	MOV	D,L	80	ADD	B	AB	XRA	E	D6	SUI	D8			

Adr. = 16-Bit-Adresse, (8) = 8-Bit-Datenbyte, (16) = 16-Bit-Datenwort

Abb. 1.2.5.2 Operationscodes und Mnemonics der CPU 8080 von Intel. Sie werden auch vom Z80 und vom 8085 ausgeführt

00	•		40	NEG	A		80	SUB	A	IMM	C0	SUB	B	IMM
01	NOP		41	•			81	CMP	A	IMM	C1	CMP	B	IMM
02	•		42	•			82	SBC	A	IMM	C2	SBC	B	IMM
03	•		43	COM	A		83	•			C3	•		
04	•		44	LSR	A		84	AND	A	IMM	C4	AND	B	IMM
05	•		45	•			85	BIT	A	IMM	C5	BIT	B	IMM
06	TAP		46	ROR	A		86	LDA	A	IMM	C6	LDA	B	IMM
07	TPA		47	ASR	A		87	•			C7	•		
08	INX		48	ASL	A		88	EOR	A	IMM	C8	EOR	B	IMM
09	DEX		49	ROL	A		89	ADC	A	IMM	C9	ADC	B	IMM
0A	CLV		4A	DEO	A		8A	ORA	A	IMM	CA	ORA	B	IMM
0B	SEV		4B	•			8B	ADD	A	IMM	CB	ADD	B	IMM
0C	CLC		4C	INC	A		8C	CPX	A	IMM	CC	•		
0D	SEC		4D	TST	A		8D	BSR		REL	CD	•		
0E	CLI		4E	•			8E	LDS		IMM	CE	LDX		IMM
0F	SEI		4F	CLR	A		8F	•			CF	•		
10	SBA		50	NEG	B		90	SUB	A	DIR	D0	SUB	B	DIR
11	CBA		51	•			91	CMP	A	DIR	D1	CMP	B	DIR
12	•		52	•			92	SBC	A	DIR	D2	SBC	B	DIR
13	•		53	COM	B		93	•			D3	•		
14	•		54	LSR	B		94	AND	A	DIR	D4	AND	B	DIR
15	•		55	•			95	BIT	A	DIR	D5	BIT	B	DIR
16	TAB		56	ROR	B		96	LDA	A	DIR	D6	LDA	B	DIR
17	TBA		57	ASR	B		97	STA	A	DIR	D7	STA	B	DIR
18	•		58	ASL	B		98	EOR	A	DIR	D8	EOR	B	DIR
19	DAA		59	ROL	B		99	ADC	A	DIR	D9	ADC	B	DIR
1A	•		5A	DEC	B		9A	ORA	A	DIR	DA	ORA	B	DIR
1B	ABA		5B	•			9B	ADD	A	DIR	DB	ADD	B	DIR
1C	•		5C	INC	B		9C	CPX		DIR	DC	•		
1D	•		5D	TST	B		9D	•			DD	•		
1E	•		5E	•			9E	LDS		DIR	DE	LDX		DIR
1F	•		5F	CLR	B		9F	STS		DIR	DF	STX		DIR
20	BRA	REL	60	NEG		IND	A0	SUB	A	IND	E0	SUB	B	IND
21	•		61	•			A1	CMP	A	IND	E1	CMP	B	IND
22	BHI	REL	62	•			A2	SBC	A	IND	E2	SBC	B	IND
23	BLS	REL	63	COM		IND	A3	•			E3	•		
24	BCC	REL	64	LSR		IND	A4	AND	A	IND	E4	AND	B	IND
25	BCS	REL	65	•			A5	BIT	A	IND	E5	BIT	B	IND
26	BNE	REL	66	ROR		IND	A6	LDA	A	IND	E6	LDA	B	IND
27	BEQ	REL	67	ASR		IND	A7	STA	A	IND	E7	STA	B	IND
28	BVC	REL	68	ASL		IND	A8	EOR	A	IND	E8	EOR	B	IND
29	BVS	REL	69	ROL		IND	A9	ADC	A	IND	E9	ADC	B	IND
2A	BPL	REL	6A	DEC		IND	AA	ORA	A	IND	EA	ORA	B	IND
2B	BMI	REL	6B	•			AB	ADD	A	IND	EB	ADD	B	IND
2C	BGE	REL	6C	INC		IND	AC	CPX		IND	EC	•		
2D	BLT	REL	6D	TST		IND	AD	JSR		IND	ED	•		
2E	BGT	REL	6E	JMP		IND	AE	LDS		IND	EE	LDX		IND
2F	BLE	REL	6F	CLR		IND	AF	STS		IND	EF	STX		IND
30	TSX		70	NEG		EXT	B0	SUB	A	EXT	F0	SUB	B	EXT
31	INS		71	•			B1	CMP	A	EXT	F1	CMP	B	EXT
32	PUL	A	72	•			B2	SBC	A	EXT	F2	SBC	B	EXT
33	PUL	B	73	COM		EXT	B3	•			F3	•		
34	DES		74	LSR		EXT	B4	AND	A	EXT	F4	AND	B	EXT
35	TXS		75	•			B5	BIT	A	EXT	F5	BIT	B	EXT
36	PSH	A	76	ROR		EXT	B6	LDA	A	EXT	F6	LDA	B	EXT
37	PSH	B	77	ASR		EXT	B7	STA	A	EXT	F7	STA	B	EXT
38	•		78	ASL		EXT	B8	EOR	A	EXT	F8	ADC	B	EXT
39	RTS		79	ROL		EXT	B9	ADC	A	EXT	F9	ADC	B	EXT
3A	•		7A	DEC		EXT	BA	ORA	A	EXT	FA	ORA	B	EXT
3B	RTI		7B	•			BB	ADD	A	EXT	FB	ADD	B	EXT
3C	•		7C	INC		EXT	BC	CPX		EXT	FC	•		
3D	•		7D	TST		EXT	BD	JSR		EXT	FD	•		
3E	WAI		7E	JMP		EXT	BE	LDS		EXT	FE	LDX		EXT
3F	SWI		7F	CLR		EXT	BF	STS		EXT	FF	STX		EXT

A = Accumulator A, B = Accumulator B, REL = Relative, IND = Index, IMM = Immediate,
DIR = Direct = Seite Null, EXT = Extended = Absolut

Abb. 1.2.5.3 Operationscodes und Mnemonics des Motorola-Prozessors 6800

OP	Mnem	Mode	~	#	OP	Mnem	Mode	~	#	OP	Mnem	Mode	~	#
00	NEG	Direct	6	2	30	LEAX	Indexed	4+	2+	60	NEG	Indexed	6+	2+
01	•	↑			31	LEAY	↑	4+	2+	61	•	↑		
02	•				32	LEAS	↓	4+	2+	62	•			
03	COM		6	2	33	LEAU	Indexed	4+	2+	63	COM		6+	2+
04	LSR		6	2	34	PSHS	Immed	5+	2	64	LSR		6+	2+
05	•				35	PULS	Immed	5+	2	65	•			
06	ROR		6	2	36	PSHU	Immed	5+	2	66	ROR		6+	2+
07	ASR		6	2	37	PULU	Immed	5+	2	67	ASR		6+	2+
08	ASL, LSL		6	2	38	•	–			68	ASL, LSL		6+	2+
09	ROL		6	2	39	RTS	Inherent	5	1	69	ROL		6+	2+
0A	DEC		6	2	3A	ABX	↑	3	1	6A	DEC		6+	2+
0B	•				3B	RTI		6/15	1	6B	•			
0C	INC		6	2	3C	CWAI	↓	≥20	2	6C	INC		6+	2+
0D	TST		6	2	3D	MUL	Inherent	11	1	6D	TST		6+	2+
0E	JMP		3	2	3E	•	–			6E	JMP		3+	2+
0F	CLR	Direct	6	2	3F	SWI	Inherent	19	1	6F	CLR	Indexed	6+	2+
10	Page 2	–	–	–	40	NEGA	Inherent	2	1	70	NEG	Extended	7	3
11	Page 3	–	–	–	41	•	↑			71	•	↑		
12	NOP	Inherent	2	1	42	•				72	•			
13	SYNC	Inherent	≥4	1	43	COMA		2	1	73	COM		7	3
14	•				44	LSRA		2	1	74	LSR		7	3
15	•				45	•				75	•			
16	LBRA	Relative	5	3	46	RORA		2	1	76	ROR		7	3
17	LBSR	Relative	9	3	47	ASRA		2	1	77	ASR		7	3
18	•				48	ASLA, LSLA		2	1	78	ASL, LSL		7	3
19	DAA	Inherent	2	1	49	ROLA		2	1	79	ROL		7	3
1A	ORCC	Immed	3	2	4A	DECA		2	1	7A	DEC		7	3
1B	•	–			4B	•				7B	•			
1C	ANDCC	Immed	3	2	4C	INCA		2	1	7C	INC		7	3
1D	SEX	Inherent	2	1	4D	TSTA		2	1	7D	TST		7	3
1E	EXG	Immed	8	2	4E	•				7E	JMP		4	3
1F	TFR	Immed	6	2	4F	CLRA	Inherent	2	1	7F	CLR	Extended	7	3
20	BRA	Relative	3	2	50	NEGB	Inherent	2	1	80	SUBA	Immed	2	2
21	BRN	↑	3	2	51	•	↑			81	CMPA	↑	2	2
22	BHI		3	2	52	•				82	SBCA		2	2
23	BLS		3	2	53	COMB		2	1	83	SUBD		4	3
24	BHS, BCC		3	2	54	LSRB		2	1	84	ANDA		2	2
25	BLO, BCS		3	2	55	•				85	BITA		2	2
26	BNE		3	2	56	RORB		2	1	86	LDA		2	2
27	BEQ		3	2	57	ASRB		2	1	87	•			
28	BVC		3	2	58	ASLB, LSLB		2	1	88	EORA		2	2
29	BVS		3	2	59	ROLB		2	1	89	ADCA		2	2
2A	BPL		3	2	5A	DECB		2	1	8A	ORA		2	2
2B	BMI		3	2	5B	•				8B	ADDA		2	2
2C	BGE		3	2	5C	INCB		2	1	8C	CMPX	Immed	4	3
2D	BLT		3	2	5D	TSTB		2	1	8D	BSR	Relative	7	2
2E	BGT		3	2	5E	•				8E	LDX	Immed	3	3
2F	BLE	Relative	3	2	5F	CLRB	Inherent	2	1	8F	•			

LEGEND:
~ Number of MPU cycles (less possible push pull or indexed-mode cycles)
# Number of program bytes
• Denotes unused opcode

Abb. 1.2.5.4 Beim 6809 gibt es wie beim Z80 einige 2-Byte-Operationscodes (siehe auch Abb. 1.2.5.7)

OP	Mnem	Mode	~	#
90	SUBA	Direct	4	2
91	CMPA	↑	4	2
92	SBCA		4	2
93	SUBD		6	2
94	ANDA		4	2
95	BITA		4	2
96	LDA		4	2
97	STA		4	2
98	EORA		4	2
99	ADCA		4	2
9A	ORA		4	2
9B	ADDA		4	2
9C	CMPX		6	2
9D	JSR		7	2
9E	LDX	↓	5	2
9F	STX	Direct	5	2
A0	SUBA	Indexed	4+	2+
A1	CMPA	↑	4+	2+
A2	SBCA		4+	2+
A3	SUBD		6+	2+
A4	ANDA		4+	2+
A5	BITA		4+	2+
A6	LDA		4+	2+
A7	STA		4+	2+
A8	EORA		4+	2+
A9	ADCA		4+	2+
AA	ORA		4+	2+
AB	ADDA		4+	2+
AC	CMPX		6+	2+
AD	JSR		7+	2+
AE	LDX	↓	5+	2+
AF	STX	Indexed	5+	2+
B0	SUBA	Extended	5	3
B1	CMPA	↑	5	3
B2	SBCA		5	3
B3	SUBD		7	3
B4	ANDA		5	3
B5	BITA		5	3
B6	LDA		5	3
B7	STA		5	3
B8	EORA		5	3
B9	ADCA		5	3
BA	ORA		5	3
BB	ADDA		5	3
BC	CMPX		7	3
BD	JSR		8	3
BE	LDX	↓	6	3
BF	STX	Extended	6	3

NOTE: All unused opcodes are both undefined and illegal

OP	Mnem	Mode	~	#
C0	SUBB	Immed	2	2
C1	CMPB	↑	2	2
C2	SBCB		2	2
C3	ADDD		4	3
C4	ANDB		2	2
C5	BITB	Immed	2	2
C6	LDB	Immed	2	2
C7	*	↑		
C8	EORB		2	2
C9	ADCB		2	2
CA	ORB		2	2
CB	ADDB		2	2
CC	LDD		3	3
CD	*	↓		
CE	LDU	Immed	3	3
CF	*			
D0	SUBB	Direct	4	2
D1	CMPB	↑	4	2
D2	SBCB		4	2
D3	ADDD		6	2
D4	ANDB		4	2
D5	BITB		4	2
D6	LDB		4	2
D7	STB		4	2
D8	EORB		4	2
D9	ADCB		4	2
DA	ORB		4	2
DB	ADDB		4	2
DC	LDD		5	2
DD	STD		5	2
DE	LDU	↓	5	2
DF	STU	Direct	5	2
E0	SUBB	Indexed	4+	2+
E1	CMPB	↑	4+	2+
E2	SBCB		4+	2+
E3	ADDD		6+	2+
E4	ANDB		4+	2+
E5	BITB		4+	2+
E6	LDB		4+	2+
E7	STB		4+	2+
E8	EORB		4+	2+
E9	ADCB		4+	2+
EA	ORB		4+	2+
EB	ADDB		4+	2+
EC	LDD		5+	2+
ED	STD		5+	2+
EE	LDU	↓	5+	2+
EF	STU	Indexed	5+	2+
F0	SUBB	Extended	5	3
F1	CMPB	↑	5	3
F2	SBCB		5	3
F3	ADDD		7	3
F4	ANDB		5	3
F5	BITB		5	3
F6	LDB		5	3
F7	STB		5	3
F8	EORB		5	3
F9	ADCB		5	3
FA	ORB		5	3
FB	ADDB	Extended	5	3
FC	LDD	Extended	6	3
FD	STD	↑	6	3
FE	LDU	↓	6	3
FF	STU	Extended	6	3

OP	Mnem	Mode	~	#
	Page 2 and 3 Machine Codes			
1021	LBRN	Relative	5	4
1022	LBHI	↑	5(6)	4
1023	LBLS		5(6)	4
1024	LBHS, LBCC		5(6)	4
1025	LBCS, LBLO		5(6)	4
1026	LBNE		5(6)	4
1027	LBEQ		5(6)	4
1028	LBVC		5(6)	4
1029	LBVS		5(6)	4
102A	LBPL		5(6)	4
102B	LBMI		5(6)	4
102C	LBGE		5(6)	4
102D	LBLT		5(6)	4
102E	LBGT		5(6)	4
102F	LBLE	Relative	5(6)	4
103F	SWI2	Inherent	20	2
1083	CMPD	Immed	5	4
108C	CMPY	↑	5	4
108E	LDY	Immed	4	4
1093	CMPD	Direct	7	3
109C	CMPY	↑	7	3
109E	LDY		6	3
109F	STY	Direct	6	3
10A3	CMPD	Indexed	7+	3+
10AC	CMPY	↑	7+	3+
10AE	LDY		6+	3+
10AF	STY	Indexed	6+	3+
10B3	CMPD	Extended	8	4
10BC	CMPY	↑	8	4
10BE	LDY		7	4
10BF	STY	Extended	7	4
10CE	LDS	Immed	4	4
10DE	LDS	Direct	6	3
10DF	STS	Direct	6	3
10EE	LDS	Indexed	6+	3+
10EF	STS	Indexed	6+	3+
10FE	LDS	Extended	7	4
10FF	STS	Extended	7	4
113F	SWI3	Inherent	20	2
1183	CMPU	Immed	5	4
118C	CMPS	Immed	5	4
1193	CMPU	Direct	7	3
119C	CMPS	Direct	7	3
11A3	CMPU	Indexed	7+	3+
11AC	CMPS	Indexed	7+	3+
11B3	CMPU	Extended	8	4
11BC	CMPS	Extended	8	4

zu Abb. 1.2.5.4

R6500 MICROPROCESSOR INSTRUCTION SET – ALPHABETIC SEQUENCE

ADC	Add Memory to Accumulator with Carry
AND	"AND" Memory with Accumulator
ASL	Shift Left One Bit (Memory or Accumulator)
BCC	Branch on Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Result Zero
BIT	Test Bits in Memory with Accumulator
BMI	Branch on Result Minus
BNE	Branch on Result not Zero
BPL	Branch on Result Plus
BRK	Force Break
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	"Exclusive-Or" Memory with Accumulator
INC	Increment Memory by One
INX	Increment Index X by One
INY	Increment Index Y by One
JMP	Jump to New Location
JSR	Jump to New Location Saving Return Address
LDA	Load Accumulator with Memory
LDX	Load Index X with Memory
LDY	Load Index Y with Memory
LSR	Shift Right One Bit (Memory or Accumulator)
NOP	No Operation
ORA	"OR" Memory with Accumulator
PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
ROL	Rotate One Bit Left (Memory or Accumulator)
ROR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return from Interrupt
RTS	Return from Subroutine
SBC	Subtract Memory from Accumulator with Borrow
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Status
STA	Store Accumulator in Memory
STX	Store Index X in Memory
STY	Store Index Y in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TSX	Transfer Stack Pointer to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Pointer
TYA	Transfer Index Y to Accumulator

Abb. 1.2.5.5 Mnemonics der NMOS-Prozessorfamilie 65XX

MNE-MONIC	OPERATION	IMMEDIATE OP	n	#	ABSOLUTE OP	n	#	ZERO PAGE OP	n	#	IMPLIED OP	n	#	(IND, X) OP	n	#	(IND), Y OP	n	#	ZPG,X OP	n	#	ABS,X OP	n	#	ABS,Y OP	n	#	RELATIVE OP	n	#	INDIRECT OP	n	#	ZPG,Y OP	n	#	PROCESSOR STATUS CODE 7 6 4 3 2 1 0 N V B D I Z C	MNE-MONIC
ADC	A + M + C → A (1,3)	69	2	2	6D	4	3	65	3	2				61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3				72	5	2				N V • • • Z C	ADC
AND	A∧M → A (1)	29	2	2	2D	4	3	25	3	2				21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3				32	5	2				N • • • • Z •	AND
ASL	C ← [7 0] ←0				0E	6	3	06	5	2	0A	2	1							16	6	2	1E	7	3													N • • • • Z C	ASL
BCC	BRANCH IF C=0 (2)																												90	2	2							• • • • • • •	BCC
BCS	BRANCH IF C=1 (2)																												B0	2	2							• • • • • • •	BCS
BEQ	BRANCH IF Z=1 (2)																												F0	2	2							• • • • • • •	BEQ
BIT	A ∧ M	89	2	2	2C	4	3	24	3	2										34	4	2	3C	4	3													M7 M6 • • Z •	BIT
BMI	BRANCH IF N=1 (2)																												30	2	2							• • • • • • •	BMI
BNE	BRANCH IF Z=0 (2)																												D0	2	2							• • • • • • •	BNE
BPL	BRANCH IF N=0 (2)																												10	2	2							• • • • • • •	BPL
BRA	BRANCH ALWAYS (2)																												80	2	2							• • • • • • •	BRA
BRK	BREAK										00	7	1																									• • 1 0 1 • •	BRK
BVC	BRANCH IF V=0 (2)																												50	2	2							• • • • • • •	BVC
BVS	BRANCH IF V=1 (2)																												70	2	2							• • • • • • •	BVS
CLC	0 → C										18	2	1																									• • • • • • 0	CLC
CLD	0 → D										D8	2	1																									• • • 0 • • •	CLD
CLI	0 → I										58	2	1																									• • • • 0 • •	CLI
CLV	0 → V										B8	2	1																									• 0 • • • • •	CLV
CMP	A-M (1)	C9	2	2	CD	4	3	C5	3	2				C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3				D2	5	2				N • • • • Z C	CMP
CPX	X-M	E0	2	2	EC	4	3	E4	3	2																												N • • • • Z C	CPX
CPY	Y-M	C0	2	2	CC	4	3	C4	3	2																												N • • • • Z C	CPY
DEC	DECREMENT				CE	6	3	C6	5	2	3A	2	1							D6	6	2	DE	7	3													N • • • • Z •	DEC
DEX	X-1 → X										CA	2	1																									N • • • • Z •	DEX
DEY	Y-1 → Y										88	2	1																									N • • • • Z •	DEY
EOR	A ∀ M →A (1)	49	2	2	4D	4	3	45	3	2				41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3				52	5	2				N • • • • Z •	EOR
INC	INCREMENT				EE	6	3	E6	5	2	1A	2	1							F6	6	2	FE	7	3													N • • • • Z •	INC
INX	X + 1 → X										E8	2	1																									N • • • • Z •	INX
INY	Y + 1 → Y										C8	2	1																									N • • • • Z •	INY
JMP	JUMP TO NEW LOC				4C	3	3							7C	6	3																6C	6	3				• • • • • • •	JMP
JSR	JUMP SUB				20	6	3																															• • • • • • •	JSR
LDA	M → A (1)	A9	2	2	AD	4	3	A5	3	2				A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3				B2	5	2				N • • • • Z •	LDA
LDX	M → X (1)	A2	2	2	AE	4	3	A6	3	2																BE	4	3							B6	4	2	N • • • • Z •	LDX
LDY	M → Y (1)	A0	2	2	AC	4	3	A4	3	2										B4	4	2	BC	4	3													N • • • • Z •	LDY
LSR	0→ [7 0] → C				4E	6	3	46	5	2	4A	2	1							56	6	2	5E	7	3													0 • • • • Z C	LSR
NOP	NO OPERATION										EA	2	1																									• • • • • • •	NOP
ORA	AVM → A (1)	09	2	2	0D	4	3	05	3	2				01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3				12	5	2				N • • • • Z •	ORA

Abb. 1.2.5.6 Operationscode-Tabelle des CMOS-Prozessors 65SC02 von GTE Microcircuits. (Vorsicht: Die Rockwell-Version weicht bei manchen gegenüber der NMOS-Version hinzugekommenen Codes hiervon ab!)

MNE-MONIC	OPERATION	IMMEDIATE OP	n	#	ABSOLUTE OP	n	#	ZERO PAGE OP	n	#	IMPLIED OP	n	#	(IND, X) OP	n	#	(IND), Y OP	n	#	ZPG,X OP	n	#	ABS,X OP	n	#	ABS,Y OP	n	#	RELATIVE OP	n	#	INDIRECT OP	n	#	ZPG,Y OP	n	#	PROCESSOR STATUS CODE 7 6 4 3 2 1 0 N V B D I Z C	MNE-MONIC
PHA	A→Ms S-1→S										48	3	1																							• • • • • • •	PHA
PHP	P→Ms S-1→S										08	3	1																							• • • • • • •	PHP
PHX	X→Ms S-1→S										DA	3	1																							• • • • • • •	PHX
PHY	Y→Ms S-1→S										5A	3	1																							• • • • • • •	PHY
PLA	S+1→S Ms→A										68	4	1																							N • • • • Z •	PLA
PLP	S+1→S Ms→P										28	4	1																							N V 1 D I Z C	PLP
PLX	S+1→S Ms→X										FA	4	1																							N Z • • • • •	PLX
PLY	S+1→S Ms→Y										7A	4	1																							N Z • • • • •	PLY
ROL	←[7 0]←C				2E	6	3	26	5	2	2A	2	1							36	6	2	3E	7	3											N • • • • Z C	ROL
ROR	→C→[7 0]				6E	6	3	66	5	2	6A	2	1							76	6	2	7E	7	3											N • • • • Z C	ROR
RTI	RTRN INT										40	6	1																							N V 1 D I Z C	RTI
RTS	RTRN SUB										60	6	1																							• • • • • • •	RTS
SBC	A-M-C̄→A (1,3)	E9	2	2	ED	4	3	E5	3	2				E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3				F2	5	2				N V • • • Z C	SBC
SEC	1 → C										38	2	1																							• • • • • • 1	SEC
SED	1 → D										F8	2	1																							• • • 1 • • •	SED
SEI	1 → I										78	2	1																							• • • • 1 • •	SEI
STA	A → M				8D	4	3	85	3	2				81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3				92	5	2				• • • • • • •	STA
STX	X → M				8E	4	3	86	3	2																									96	4	2	• • • • • • •	STX
STY	Y → M				8C	4	3	84	3	2										94	4	2																• • • • • • •	STY
STZ	00 → M				9C	4	3	64	3	2										74	4	2	9E	5	3													• • • • • • •	STZ
TAX	A → X										AA	2	1																							N • • • • Z •	TAX
TAY	A → Y										A8	2	1																							N • • • • Z •	TAY
TRB	A̅∧M → M				1C	6	3	14	5	2																											• • • • • Z •	TRB
TSB	A∨M → M				0C	6	3	04	5	2																											• • • • • Z •	TSB
TSX	S → X										BA	2	1																							N • • • • Z •	TSX
TXA	X → A										8A	2	1																							N • • • • Z •	TXA
TXS	X → S										9A	2	1																							• • • • • • •	TXS
TYA	Y → A										98	2	1																							N • • • • Z •	TYA

Notes:

1. Add 1 to "n" if page boundary is crossed.
2. Add 1 to "n" if branch occurs to same page. Add 2 to "n" if branch occurs to different page.
3. Add 1 to "n" if decimal mode.

X Index X
Y Index Y
A Accumulator
M Memory per effective address
Ms Memory per stack pointer

\+ Add
\- Subtract
∧ And
∨ Or
∨̶ Exclusive or

n No. Cycles
No. Bytes
M6 Memory Bit #6
M7 Memory Bit #7

zu Abb. 1.2.5.6

8-BIT ACCUMULATOR AND MEMORY INSTRUCTIONS

Mnemonic(s)	Operation
ADCA, ADCB	Add memory to accumulator with carry
ADDA, ADDB	Add memory to accumulator
ANDA, ANDB	And memory with accumulator
ASL, ASLA, ASLB	Arithmetic shift of accumulator or memory left
ASR, ASRA, ASRB	Arithmetic shift of accumulator or memory right
BITA, BITB	Bit test memory with accumulator
CLR, CLRA, CLRB	Clear accumulator or memory location
CMPA, CMPB	Compare memory from accumulator
COM, COMA, COMB	Complement accumulator or memory location
DAA	Decimal adjust A accumulator
DEC, DECA, DECB	Decrement accumulator or memory location
EORA, EORB	Exclusive or memory with accumulator
EXG R1, R2	Exchange R1 with R2 (R1, R2 = A, B, CC, DP)
INC, INCA, INCB	Increment accumulator or memory location
LDA, LDB	Load accumulator from memory
LSL, LSLA, LSLB	Logical shift left accumulator or memory location
LSR, LSRA, LSRB	Logical shift right accumulator or memory location
MUL	Unsigned multiply (A × B — D)
NEG, NEGA, NEGB	Negate accumulator or memory
ORA, ORB	Or memory with accumulator
ROL, ROLA, ROLB	Rotate accumulator or memory left
ROR, RORA, RORB	Rotate accumulator or memory right
SBCA, SBCB	Subtract memory from accumulator with borrow
STA, STB	Store accumulator to memory
SUBA, SUBB	Subtract memory from accumulator
TST, TSTA, TSTB	Test accumulator or memory location
TFR R1, R2	Transfer R1 to R2 (R1, R2 = A, B, CC, DP)

NOTE: A, B, CC, or DP may be pushed to (pulled from) stack with either PSHS, PSHU (PULS, PULU) instructions.

16-BIT ACCUMULATOR AND MEMORY INSTRUCTIONS

Mnemonic(s)	Operation
ADDD	Add memory to D accumulator
CMPD	Compare memory from D accumulator
EXG D, R	Exchange D with X, Y, S, U, or PC
LDD	Load D accumulator from memory
SEX	Sign Extend B accumulator into A accumulator
STD	Store D accumulator to memory
SUBD	Subtract memory from D accumulator
TFR D, R	Transfer D to X, Y, S, U, or PC
TFR R, D	Transfer X, Y, S, U, or PC to D

NOTE: D may be pushed (pulled) to stack with either PSHS, PSHU (PULS, PULU) instructions.

Abb. 1.2.5.7 Mnemonics des Prozessors 6809 von Motorola (siehe auch Abb. 1.2.5.4)

INDEX REGISTER/STACK POINTER INSTRUCTIONS

Instruction	Description
CMPS, CMPU	Compare memory from stack pointer
CMPX, CMPY	Compare memory from index register
EXG R1, R2	Exchange D, X, Y, X, U, or PC with D, X Y, S, U, or PC
LEAS, LEAU	Load effective address into stack pointer
LEAX, LEAY	Load effective address into index register
LDS, LDU	Load stack pointer from memory
LDX, LDY	Load index register from memory
PSHS	Push A, B, CC, DP, D, X, Y, U, or PC onto hardware stack
PSHU	Push A, B, CC, DP, D, X, Y, S, or PC onto user stack
PULS	Pull A, B, CC, DP, D, X, Y, U, or PC from hardware stack
PULU	Pull A, B, CC, DP, D, X, Y, S, or PC from hardware stack
STS, STU	Store stack pointer to memory
STX, STY	Store index register to memory
TFR R1, R2	Transfer D, X, Y, S, U or PC to D, X, Y, S, U, or PC
ABX	Add B accumulator to X (unsigned)

BRANCH INSTRUCTIONS

Instruction	Description
SIMPLE BRANCHES	
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BMI, LBMI	Branch if minus
BPL, LBPL	Branch if plus
BCS, LBCS	Branch if carry set
BCC, LBCC	Branch if carry clear
BVS, LBVS	Branch if overflow set
BVC, LBVC	Branch if overflow clear
SIGNED BRANCHES	
BGT, LBGT	Branch if greater (signed)
BVS, LBVS	Branch if invalid 2s complement result
BGE, LBGE	Branch if greater than or equal (signed)
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BLE, LBLE	Branch if less than or equal (signed)
BVC, LBVC	Branch if valid 2s complement result
BLT, LBLT	Branch if less than (signed)
UNSIGNED BRANCHES	
BHI, LBHI	Branch if higher (unsigned)
BCC, LBCC	Branch if higher or same (unsigned)
BHS, LBHS	Branch if higher or same (unsigned)
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BLS, LBLS	Branch if lower or same (unsigned)
BCS, LBCS	Branch if lower (unsigned)
BLO, LBLO	Branch if lower (unsigned)
OTHER BRANCHES	
BSR, LBSR	Branch to subroutine
BRA, LBRA	Branch always
BRN, LBRN	Branch never

zu Abb. 1.2.5.7

MISCELLANEOUS INSTRUCTIONS

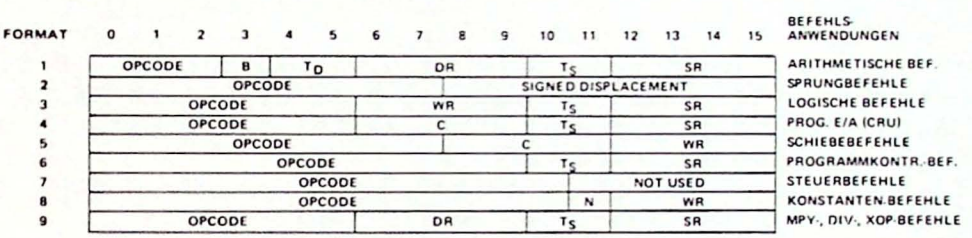
Instruction	Description
ANDCC	AND condition code register
CWAI	AND condition code register, then wait for interrupt
NOP	No operation
ORCC	OR condition code register
JMP	Jump
JSR	Jump to subroutine
RTI	Return from interrupt
RTS	Return from subroutine
SWI, SWI2, SWI3	Software interrupt (absolute indirect)
SYNC	Synchronize with interrupt line

zu Abb. 1.2.5.7

Maschinen-sprache OP-Code (Hexadezimal)	Assembler-sprache Mnemonics	Befehlsbeschreibung	Format	Beeinflußte Statusbits
0200	LI	Load Immediate	8	0-2
0220	AI	Add Immediate	8	0-4
0240	ANDI	And Immediate	8	0-2
0260	ORI	Or Immediate	8	0-2
0280	CI	Compare Immediate	8	0-2
02A0	STWP	Store WP	8	–
02C0	STST	Store ST	8	–
02E0	LWPI	Load WP immediate	8	–
0300	LIMI	Load Int. Mask	8	12-15
0340	IDLE	Idle	7	–
0360	RSET	Reset AU	7	12-15
0380	RTWP	Return from Context Sw.	7	0-15
03A0	CKON	User Defined	7	–
03C0	CKOF	User Defined	7	–
03E0	LREX	Load & Execute	7	–
0400	BLWP	Branch; New WP	6	–
0440	B	Branch	6	–
0480	X	Execute	6	–
04C0	CLR	Clear to Zeroes	6	–
0500	NEG	Negate to Ones	6	0-2
0540	INV	Invert	6	0-2
0580	INC	Increment by 1	6	0-4
05C0	INCT	Increment by 2	6	0-4
0600	DEC	Decrement by 1	6	0-4
0640	DECT	Decrement by 2	6	0-4
0680	BL	Branch and Link	6	–
06C0	SWPB	Swap Bytes	6	–
0700	SETO	Set to Ones	6	–
0740	ABS	Absolute Value	6	0-2
0800	SRA	Shift Right Arithmetic	5	0-3
0900	SRL	Shift Right Logical	5	0-3
0A00	SLA	Shift Left Arithmetic	5	0-4
0B00	SRC	Shift Right Circular	5	0-3
1000	JMP	Unconditional Jump	2	–
1100	JLT	Jump on Less Than	2	–
1200	JLE	Jump on Less Than or Equal	2	–
1300	JEQ	Jump on Equal	2	–

Abb. 1.2.5.8 Mnemonics und Operationscodes der CPUs 9900 und 9980 von Texas Instruments

Maschinensprache OP-Code (Hexadezimal)	Assemblersprache Mnemonics	Befehlsbeschreibung	Format	Beeinflußte Statusbits
1400	JHE	Jump on High or Equal	2	–
1500	JGT	Jump on Greater Than	2	–
1600	JNE	Jump on Not Equal	2	–
1700	JNC	Jump on No Carry	2	–
1800	JOC	Jump on Carry	2	–
1900	JNO	Jump on No Overflow	2	–
1A00	JL	Jump on Low	2	–
1B00	JH	Jump on High	2	–
1C00	JOP	Jump on Odd Parity	2	–
1D00	SBO	Set CRU Bits to Ones	2	–
1E00	SBZ	Set CRU Bits to Zeroes	2	–
1F00	TB	Test CRU Bit	2	2
2000	COC	Com pare Ones Corresponding	3	2
2400	CZC	Compare Zeroes Corresponding	3	2
2800	XOR	Exclusive Or	3	0-2
2C00	XOP	Extended Operation	9	6
3000	LDCR	Load CRU	4	0-2,5
3400	STCR	Store CRU	4	0-2,5
3800	MPY	Multiply	9	–
3C00	DIV	Divide	9	4
4000	SZC	Set Zeroes Corresponding (Word)	1	0-2
5000	SZCB	Set Zeroes Corresponding (Byte)	1	0-2,5
6000	S	Subtract Word	1	0-4
7000	SB	Subtract Byte	1	0-5
8000	C	Compare Word	1	0-2
9000	CB	Compare Byte	1	0-2,5
A000	A	Add Word	1	0-4
B000	AB	Add Byte	1	0-5
C000	MOV	Move Word	1	0-2
D000	MOVB	Move Byte	1	0-2,5
E000	SOC	Set Ones Corresponding (Word)	1	0-2
F000	SOCB	Set Ones Corresponding (Byte)	1	0-2,5



FELD	BEDEUTUNG
B	BYTE INDIKATOR (1 = BYTE)
T_D	DATENSENKENADRESSTYP*
DR	DATENSENKENREGISTER
T_S	DATENQUELLENADRESSTYP*
SR	DATENQUELLENREGISTER
C	E/A- ODER SCHIEBEFORMAT (1 bis 16 Bit)
WR	ARBEITSREGISTER
N	NICHT BENUTZT (NOT USED)

*T_D ODER T_S

00	REGISTER
01	REGISTER INDIREKT
10	DIREKT ODER SYMBOLISCH (SR ODER DR = 0) INDEX (SR ODER DR > 0)
11	REGISTER INDIREKT MIT AUTO-INKREMENT

zu Abb. 1.2.5.8

```
8086/8088-Mnemonics. Verwendete Kürzel:
Q=Quelle (Reg.,Adr.,Wert), Z=Ziel (Reg.,Adr.), i=Bitanzahl.


AAA             ASCII-Abgleich für Addition
AAD             ASCII-Abgleich für Division
ADC Z,Q         Addition mit Carry
ADD Z,Q         Addition (ohne Übertrag)
AND Z,Q         Und-Verknüpfung

CALL Z          Unterprogramm-Aufruf
CBW             Konversion Byte zu Wort
CLC             Carry-Flag löschen
CLD             Direction-Flag löschen
CLI             Interrupt-Flag löschen (d.h.Interr.sperren)
CMC             Carry-Flag invertieren
CMP Z,Q         Vergleich (compare)
CMPS Z,Q        String-Vergleich (compare strings)
CWD             Konversion Wort zu Doppelwort

DAA             Dezimal-Abgleich für Addition
DAS             Dezimal-Abgleich für Subtraktion
DEC Z           Um 1 dekrementieren
DIV Q           Division ohne Vorzeichen (AX/Q), Erg.in AX,DX

ESC             Escape; Coprozessor-Opcode folgt

HLT             Halt bis zum nächsten Reset oder Interrupt

IDIV Q          Integer-Division AX/Q mit Vorzeichen, Erg.AX/DX
IMUL Q          Integer-Multiplikation AX*Q, Erg.in AX,DX
IN AX,Q         Port-Abfrage; statt AX bei 8 Bit auch AL mögl.
INC Z           Um 1 inkrementieren
INT Z           Interrupt auslösen
INTO            Interrupt bei Overflow auslösen
IRET            Rückkehr aus Interrupt-Routine

JA, JNBE Z      Sprung, wenn darüber (nach CMP)
JAE, JNB Z      Sprung, wenn darüber oder gleich (nach CMP)
JB, JNAE Z      Sprung, wenn darunter (nach CMP)
JBE, JNA Z      Sprung, wenn darunter oder gleich (nach CMP)
JC Z            Sprung, wenn Carry gesetzt
JCZX Z          Sprung, wenn CX-Register Null enthält
JE, JZ Z        Sprung, wenn gleich (nach CMP) oder Null
```

Abb. 1.2.5.9 Mnemonics der CPUs 8086 und 8088 von Intel

```
JG, JNLE Z      Sprung, wenn größer bzw. nicht kleiner/gleich
JGE, JNL Z      Sprung, wenn größer oder gleich (nach CMP)
JL, JNGE Z      Sprung, wenn kleiner bzw.nicht größer/gleich
JLE, JNE Z      Sprung, wenn kleiner oder gleich (nach CMP)
JMP Z           Jump (unbedingter Sprung)
JNC Z           Sprung, wenn Carry nicht gesetzt
JNE, JNZ Z      Sprung, wenn ungleich (nach CMP) bzw.nicht Null
JNO Z           Sprung, wenn kein Overflow
JNP, JPO Z      Sprung bei ungerader Parität
JNS Z           Sprung, wenn positiv (not sign)
JO Z            Sprung, wenn Overflow
JP, JPE Z       Sprung bei gerader Parität
JS Z            Sprung, wenn negativ (sign)


LAHF            AH mit Flags laden
LDS Z,Q         Pointer via DS laden
LEA Z,Q         Effektive Adresse laden
LES Z,Q         Pointer via ES laden
LOCK            Bus sperren (verhindert Interr.bei folg.Befehl)
LODS Q          String via SI nach AX laden, SI inkrem.
LOOP Z          CX dekrem. und Sprung nach Z, wenn nicht Null
LOOPE,LOOPZ Z   CX dekr.und Sprung nach Z, wenn >0 und Null-Flag=1
LOOPNE,LOOPNZ Z CX dekr.und Sprg. nach Z, wenn >0 und Null-Flag=0


MOV Z,Q         Inhalt kopieren (move)
MOVS Z,Q        String via SI nach Adr.in DI kopieren, SI+DI inkr.
MOVSB, MOVSW    Wie MOVS, ohne Operand, mit Unterscheidg.Byte/Wort
MUL Q           Multiplikation ohne Vorzeichen, Erg. in AX, DX


NEG Z           Negieren
NOP             Keine Wirkung (no operation); 1 Byte/3 Taktzyklen
NOT Z           Logisches "Nicht" (alle Bits invertieren)


OR Z,Q          Logische Oder-Verknüpfung
OUT Z,AX        Port-Ausgabe; statt AX auch AL möglich


POP Z           Wort vom Stack holen
POPF            Flag-Register vom Stack holen
PUSH Q          Wort auf Stack bringen
PUSHF           Flag-Register auf Stack bringen


RCL Z,i         i-mal einschl. Carry nach links rotieren
RCR Z,i         i-mal einschl. Carry nach rechts rotieren
REP             Nachfolgende Anweisung MOVS oder STOS wiederholen
REPE, REPZ      Nachf. Anw. MOVS oder STOS wiederh., wenn gleich
REPNE, REPNZ    Nachf. Anw. MOVS oder STOS wiederh., wenn ungleich
RET i           Rückkehr aus Unterpgm., optional i Worte vom Stack
```

zu Abb. 1.2.5.9

```
ROL Z,i         Um i Bits nach links rotieren
ROR Z,i         Um i Bits nach rechts rotieren


SAHF            AH-Register in Flags kopieren
SAL, SHL Z,i    Arithmetisch um i Bits nach links schieben
SAR Z,i         Arithmetisch um i Bits nach rechts schieben
SBB Z,Q         Subtraktion mit Übertrag (borrow)
SCAS Z,Q        String via DI mit AL oder AX vergl., DI inkrem.
SHR Z,i         Um i Bits nach rechts schieben
STC             Carry-Flag setzen
STD             Richtungsflag setzen (direction)
STI             Interrupts zulassen (set interrupt enable)
STOS Z          AL oder AX byte-/wortweise via DI speichern
SUB Z,Q         Subtraktion ohne Übertrag


TEST Z,Q        Bitweiser Test (Und-verknüpft)


WAIT            Warten, bis TEST-Anschluß der CPU aktiv (low)


XCHG Z,Q        Werte tauschen (exchange)
XLAT Q          Übersetzt AL mit Hilfe der 256-Byte-Tab.an Adr.BX=Q
XOR Z,Q         Logische Exklusiv-Oder-Verknüpfung
```

Hinweise:

1. Bedingte Sprünge verwenden ein Byte als relative Adresse; ihre "Reichweite" beträgt deshalb nur +/- 127 Byte.

2. Viele Anweisungen können durch Ergänzen mit einem "Segment Override Prefix" (CS:, ES: usw.) auch auf andere als das aktuelle Segment angewendet werden, z.B. MOV ES:STRG2,AL. Der Assembler generiert dann vor diesem Befehl einen Segment-Override-Opcode.

3. Als "effektive Adresse" bezeichnet man die Adresse bezogen auf den Beginn des aktuellen Segments (16-Bit-Offset-Adresse).

4. String-Befehle wie MOVS usw. inkrementieren normalerweise SI bzw. DI. Wird jedoch mit STD das Direction-Flag gesetzt, wird das Quell- bzw.Zielregister dekrementiert.

---

zu Abb. 1.2.5.9

Flag-Beeinflussung:

Nicht aufgeführte
Befehle verändern
die Flags nicht.
Abkürzungen:
- = unverändert
0 = gelöscht
1 = gesetzt
x = je n.Ergebnis
u = undefiniert
r = restauriert

Mnem.	ODITSZAPC
AAA	u---uuxux
AAD	u---xxuxu
AAM	u---xxuxu
AAS	u---uuxux
ADC	x---xxxxx
ADD	x---xxxxx
AND	0---xxux0
CLC	--------0
CLD	-0-------
CLI	--0------
CMC	--------x
CMP	x---xxxxx
CMPS	x---xxxxx
DAA	x---xxxxx
DEC	x---xxxxx
DIV	u---uuuuu
IDIV	u---uuuuu
IMUL	x---uuuux
INC	x---xxxx-
INT	--00-----
INTO	--00-----
IRET	rrrrrrrrr
MUL	x---uuuux
NEG	x---xxxx1
OR	0---xxux0
POPF	rrrrrrrrr
RCL	x-------x
ROL	x-------x
ROR	x-------x
SAHF	----rrrrr
SAL	x-------x
SAR	x---xxuxx
SBB	x---xxxxx
SCAS	x---xxxxx
SHR	x-------x
STC	--------1
STD	-1-------
STI	--1------
SUB	x---xxxxx
TEST	0---xxux0
XOR	0---xxux0

Segmentregister-Verwendung

Zugriffsart	Normales Segment	Optional m.Prefix
Programmcode	CS	-
Stackoperation	SS	-
Daten	DS	CS,ES,SS
String-Quelle	DS	CS,ES,SS
String-Ziel	ES	- (!)
BX	SS	CS,DS,ES

zu Abb. 1.2.5.9

Hi \ Lo	0	1	2	3	4	5	6	7
0	ADD b,f,r/m	ADD w,f,r/m	ADD b,t,r/m	ADD w,t,r/m	ADD b,ia	ADD w,ia	PUSH ES	POP ES
1	ADC b,f,r/m	ADC w,f,r/m	ADC b,t,r/m	ADC w,t,r/m	ADC b,i	ADC w,i	PUSH SS	POP SS
2	AND b,f,r/m	AND w,f,r/m	AND b,t,r/m	AND w,t,r/m	AND b,i	AND w,i	SEG =ES	DAA
3	XOR b,f,r/m	XOR w,f,r/m	XOR b,t,r/m	XOR w,t,r/m	XOR b,i	XOR w,i	SEG =SS	AAA
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI
6								
7	JO	JNO	JB/ JNAE	JNB/ JAE	JE/ JZ	JNE/ JNZ	JBE/ JNA	JNBE/ JA
8	Immed b,r/m	Immed w,r/m	Immed b,r/m	Immed is,r/m	TEST b,r/m	TEST w,r/m	XCHG b,r/m	XCHG w,r/m
9	XCHG AX	XCHG CX	XCHG DX	XCHG BX	XCHG SP	XCHG BP	XCHG SI	XCHG DI
A	MOV m → AL	MOV m → AX	MOV AL → m	MOV AX → m	MOVS	MOVS	CMPS	CMPS
B	MOV i → AL	MOV i → CL	MOV i → DL	MOV i → BL	MOV i → AH	MOV i → CH	MOV i → DH	MOV i → BH
C			RET, (i+SP)	RET	LES	LDS	MOV b,i,r/m	MOV w,i,r/m
D	Shift b	Shift w	Shift b,v	Shift w,v	AAM	AAD		XLAT
E	LOOPNZ/ LOOPNE	LOOPZ/ LOOPE	LOOP	JCXZ	IN b	IN w	OUT b	OUT w
F	LOCK		REP	REP z	HLT	CMC	Grp 1 b,r/m	Grp 1 w,r/m

where:

mod□r/m	000	001	010	011	100	101	110	111
Immed	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
Shift	ROL	ROR	RCL	RCR	SHL/SAL	SHR	—	SAR
Grp 1	TEST	—	NOT	NEG	MUL	IMUL	DIV	IDIV
Grp 2	INC	DEC	CALL id	CALL I,id	JMP id	JMP I,id	PUSH	—

Abb. 1.2.5.10 Befehlscodes der Prozessoren 8086 und 8088 von Intel

8	9	A	B	C	D	E	F
OR b,f,r/m	OR w,f,r/m	OR b,t,r/m	OR w,t,r/m	OR b,i	OR w,i	PUSH CS	
SBB b,f,r/m	SBB w,f,r/m	SBB b,t,r/m	SBB w,t,r/m	SBB b,i	SBB w,i	PUSH DS	POP DS
SUB b,f,r/m	SUB w,f,r/m	SUB b,t,r/m	SUB w,t,r/m	SUB b,i	SUB w,i	SEG =CS	DAS
CMP b,f,r/m	CMP w,f,r/m	CMP b,t,r/m	CMP w,t,r/m	CMP b,i	CMP w,i	SEG =DS	AAS
DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
JS	JNS	JP/ JPE	JNP/ JPO	JL/ JNGE	JNL/ JGE	JLE/ JNG	JNLE/ JG
MOV b,f,r/m	MOV w,f,r/m	MOV b,t,r/m	MOV w,t,r/m	**MOV sr,f,r/m**	LEA	**MOV sr,t,r/m**	POP r/m
CBW	CWD	CALL l,d	WAIT	PUSHF	POPF	SAHF	LAHF
TEST b,i,a	**TEST w,i,a**	**STOS**	**STOS**	**LODS**	**LODS**	**SCAS**	**SCAS**
MOV i → AX	MOV i → CX	MOV i → DX	MOV i → BX	MOV i → SP	MOV i → BP	MOV i → SI	MOV i → DI
		RET, l,(i+SP)	RET l	INT Type 3	INT (Any)	INTO	IRET
ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7
CALL d	JMP d	JMP l,d	JMP si,d	**IN v,b**	**IN v,w**	**OUT v,b**	**OUT v,w**
CLC	STC	CLI	STI	CLD	STD	Grp 2 b,r/m	Grp 2 w,r/m

b = byte operation
d = direct
f = from CPU reg
i = immediate
ia = immed. to accum.
id = indirect
is = immed. byte, sign ext.
l = long ie. intersegment
m = memory
r/m = EA is second byte
si = short intrasegment
sr = segment register
t = to CPU reg
v = variable
w = word operation
z = zero

zu Abb. 1.2.5.10

```
ABCD    Add Decimal with Extend
ADD     Add Binary
ADDA    Add Address
ADDI    Add Immediate
ADDQ    Add Quick
ADDX    Add Extended
AND     And Logical
ANDI    And Immediate
ASL     Arithmetic Shift Left
ASR     Arithmetic Shift Right

BCC     Branch Conditionally
BCHG    Test Bit and Change
BCLR    Test Bit and Clear
BRA     Branch Always
BSET    Test Bit and Set
BSR     Branch to Subroutine
BTST    Test Bit

CHK     Check Register Bounds
CLR     Clear Operand
CMP     Arithmetic Compare
CMPA    Arithm. Compare Address
CMPI    Compare Immediate
CMPM    Compare Memory

DBCC    Test, Decrement, Branch
DIVS    Signed Divide
DIVU    Unsigned Divide

EOR     Exclusive Or Logical
EORI    EOR Immediate
EXG     Exchange Registers
EXT     Sign Extend

JMP     Jump
JSR     Jump to Subroutine

LEA     Load Effective Address
LINK    Link and Allocate
LSL     Logical Shift Left
LSR     Logical Shift Right

MOVE    Move Source,Destination
        (CCR=Condition Codes, SR=Status
        Register, USP=User Stack Pointer)
```

Abb. 1.2.5.11 Mnemonics der CPU 68000 von Motorola

```
MOVEA    Move Address
MOVEM    Move Multiple Registers
MOVEP    Move Peripheral Data
MOVEQ    Move Quick
MULS     Signed Multiply
MULU     Unsigned Multiply

NBCD     Negate Decimal with Extend
NEG      Negate (2's complement)
NEGX     Negate with Extend
NOP      No Operation
NOT      Logical Complement

OR       Logical Or
ORI      Or Immediate

PEA      Push Effective Address

RESET    Reset External Devices
ROL      Rotate Left
ROR      Rotate Right
ROXL     Rotate Left with Extend
ROXR     Rotate Right with Extend
RTE      Return from Exception
RTR      Return and Restore Condition C.
RTS      Return from Subroutine

SBCD     Subtract Decimal with Extend
SCC      Set According to Condition
STOP     Stop Program Execution
SUB      Subtract binary
SUBA     Subtradt Address
SUBI     Subtract Immediate
SUBQ     Subtract Quick
SUBX     Subtract with Extend
SWAP     Swap Register Halves

TAS      Test and Set Operand
TRAP     Software Trap
TRAPV    Trap on Overflow
TST      Test Operand

UNLK     Unlink
```

zu Abb. 1.2.5.11

1.3 Halbleiterspeicher

Diejenigen Speicherzellen, auf die der Mikroprozessor direkt, d.h. über seinen Adressen- und Datenbus zugreifen kann, müssen auf Anforderung Informationen in Bruchteilen einer Mikrosekunde zur Verfügung stellen können. Dafür verwendet man integrierte Schaltungen (ICs), die heute ausschließlich aus Silizium als Grundmaterial gefertigt werden. RAM-Bausteine (Random Access Memory, Schreib-Lese-Speicher) lassen sich nach ihrem Funktionsprinzip in statische und dynamische Speicher einteilen.

1.3.1 Statische Speicher

Statische Speicher heißen so, weil ihre Logik einen statischen Zustand einnimmt, solange kein Speicherzugriff stattfindet. Sie sind in der Anwendung am problemlosesten, da das gesamte Zeitverhalten des Systems allein von der CPU bestimmt werden kann und sich die Speicherbausteine danach richten. *Abb. 1.3.1.1* zeigt zwei typische Innenschaltungen statischer Speicherchips; dabei ist jeweils nur eine Speicherzelle (1 Bit) dargestellt.

Bei der Version mit Multiemitter-Transistoren dienen die Bitleitungen BL und BL– der Zuführung der zu speichernden Daten. Zum Einschreiben liegt die Wortleitung X auf logisch 1, BL und BL– werden gegenphasig angesteuert. Es leitet immer nur einer der beiden Transistoren. Zum Lesen wird X auf logisch 0 gelegt, der Emitterstrom des jeweils leitenden Transistors fließt über X ab. Solche mit bipolaren Transistoren aufgebauten Speicherchips sind allerdings nur noch dort üblich, wo es auf extrem kleine Zugriffszeiten ankommt, da eine hohe Speicherdichte so

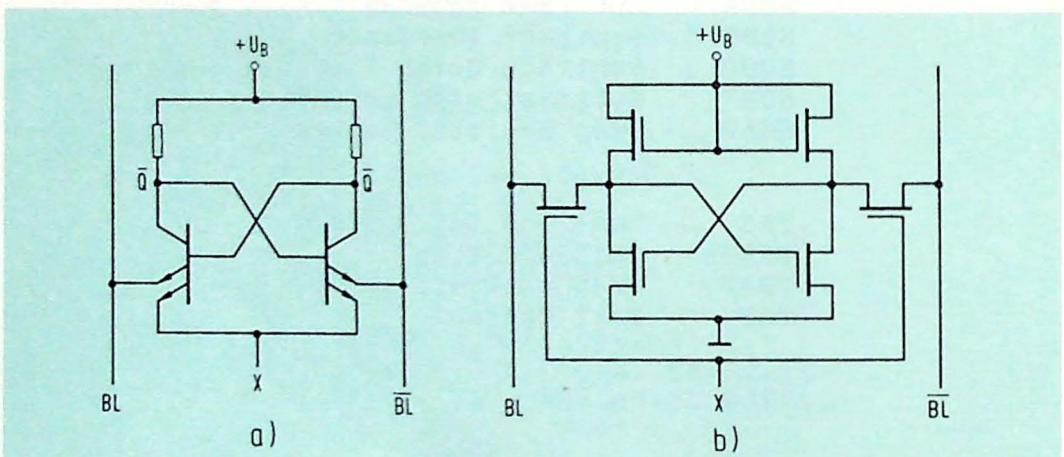


Abb. 1.3.1.1 Aufbau einer statischen Speicherzelle mit zwei bipolaren Multiemitter-Transistoren (a) und mit sechs MOS-Feldeffekt-Transistoren (b)

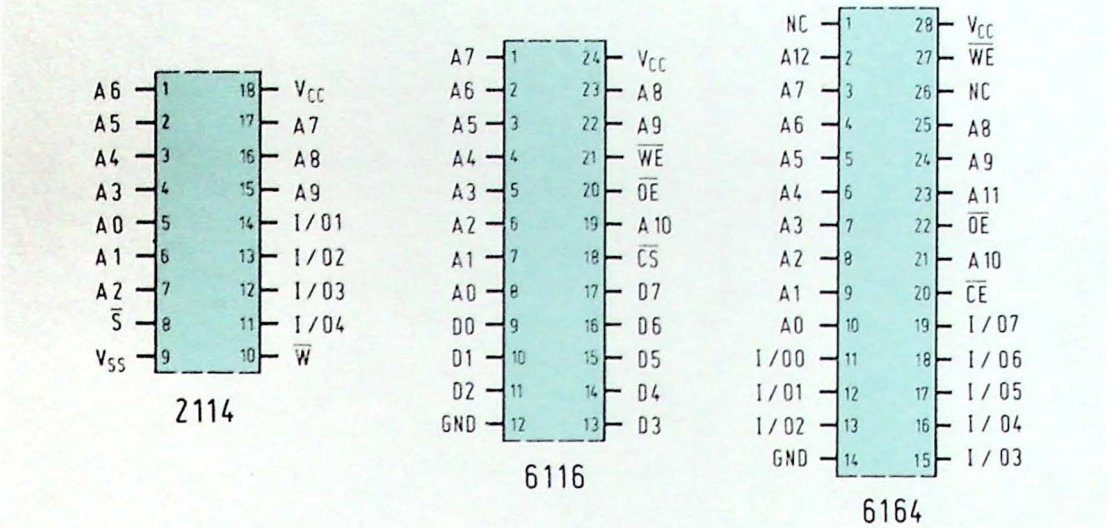


Abb. 1.3.1.2 Anschlußbilder einiger verbreiteter statischer Speicherbausteine (2114 = NMOS, 61XX = CMOS)

leider nicht erzielt werden kann. Dies gilt insbesondere auch für die sogenannte ECL-Technik, bei der emittergekoppelte Transistorpaare als Flipflops zur Speicherung dienen, womit sich Zugriffszeiten von weniger als einer Nanosekunde erzielen lassen.

Die mit MOS-Feldeffekt-Transistoren aufgebaute zweite Version wird ähnlich angesteuert. Bei X = 1 kann die Zelle über die beiden Bitleitungen gesetzt oder rückgesetzt werden. Statt Kollektor-Arbeitswiderständen sind weitere MOS-Transistoren eingebaut. Dem Nachteil der relativ großen Fläche pro Bit steht bei statischen CMOS-RAMs der Vorteil gegenüber, daß wegen der geringen Stromaufnahme die Wärmeableitung auch bei großen Chipflächen kein Problem darstellt. *Abb. 1.3.1.2* zeigt die Anschlußbilder heute üblicher MOS-Speicherchips. Sie kommen wie die meisten Mikroprozessoren mit einer einzigen Versorgungsspannung von +5 V aus. Im Ruhezustand ist der Stromverbrauch von CMOS-Speichern zu vernachlässigen, lediglich bei Speicherzugriffen schlucken sie zum Umladen interner Kapazitäten einige Milliampere.

1.3.2 Dynamische Speicher

Dynamische Speicher heißen so, weil sich in ihrem Inneren auch dann etwas tut, wenn die CPU gerade gar nichts von ihnen will. Da die Information bei ihnen nicht mit Hilfe von Flipflops, sondern durch Aufladen integrierter Kondensatoren gespeichert wird und wegen der nie idealen Sperrschicht-Isolation diese Ladung mit der Zeit verloren gehen würde, müssen sie regelmäßig „aufgefrischt" werden —

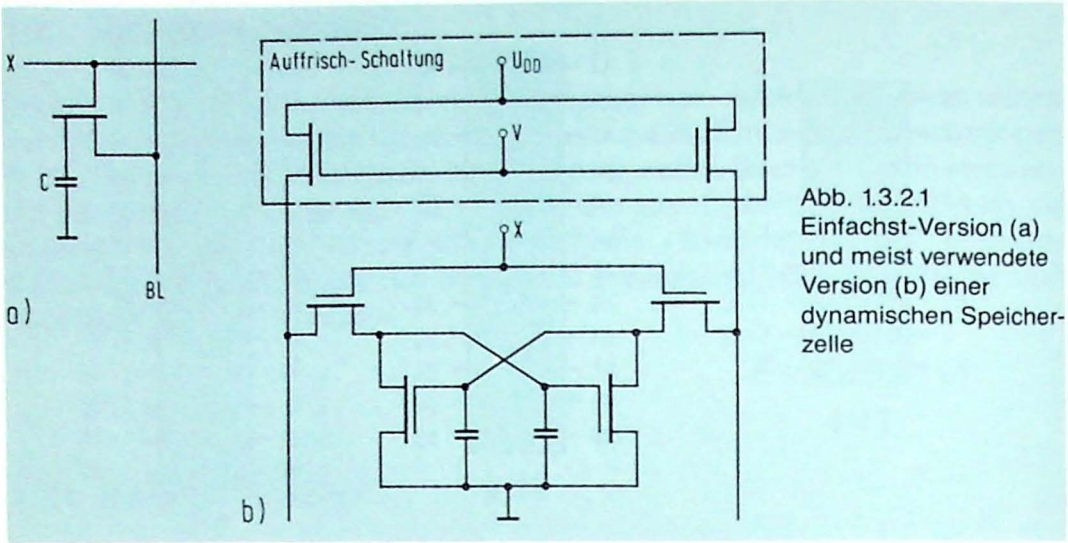


Abb. 1.3.2.1
Einfachst-Version (a) und meist verwendete Version (b) einer dynamischen Speicherzelle

nämlich etwa im Millisekunden-Rhythmus. Dafür ist meist eine externe Logik nötig, die entweder in der CPU integriert sein kann (z.B. bei Z80 und Z8000) oder die separat aufgebaut werden muß.

Abb. 1.3.2.1 zeigt unterschiedliche Möglichkeiten des internen Aufbaus dynamischer Speicherzellen. Im Extremfall kommt man sogar mit einem einzigen MOS-Transistor pro Bit aus, so daß sehr hohe Speicherdichten möglich sind. Der Kondensator wird beim Schreiben über die Bitleitung auf den gewünschten Pegel aufgeladen (0 oder 1). Beim Lesen geht leider die Kondensator-Ladung verloren. Das Auslesen wirkt also datenzerstörend. Es sind deshalb intern zusätzliche Schaltungen erforderlich, die den ursprünglichen Zustand feststellen und den Zelleninhalt auffrischen.

Die Speicherzellen sind üblicherweise matrixförmig verschaltet, damit man auf dem Chip mit möglichst wenig Querverbindungen auskommt *(Abb. 1.3.2.2)*. Die zwei Decoder sorgen dafür, daß die richtigen Zeilen- und Spaltenleitungen und damit die richtige Speicherzelle angesprochen wird, wenn eine bestimmte Adresse (A) auf dem Adressenbus liegt. Wie ein realer Speicher intern organisiert ist, zeigt *Abb. 1.3.2.3* am Beispiel des ICs 4116, einem 16 K × 1-Speicher. Das bedeutet, daß er nur ein Bit „breit" ist und 16 × 1024 = 16384 Bits speichern kann. Die Zellen sind auf 128 Zeilen und 128 Spalten aufgeteilt. Die Zeilen sind in zwei Gruppen zu je 64 angeordnet, dazwischen befindet sich der Leseverstärker.

Die Adressenleitungen A0...A6 und A7...A13 werden beim 4116 gemultiplext, d.h. bei einem Zugriff nacheinander in zwei Gruppen übertragen. Dadurch ist es möglich, mit einem preiswerten und platzsparenden 16-Pin-Gehäuse auszukommen. Die Signale CAS (Column Address Strobe = Spalten-Adressenimpuls) und RAS (Row Address Strobe = Zeilen-Adressenimpuls) dienen dazu, dem Speicherchip mitzuteilen, wann welche Adressenhälfte anliegt.

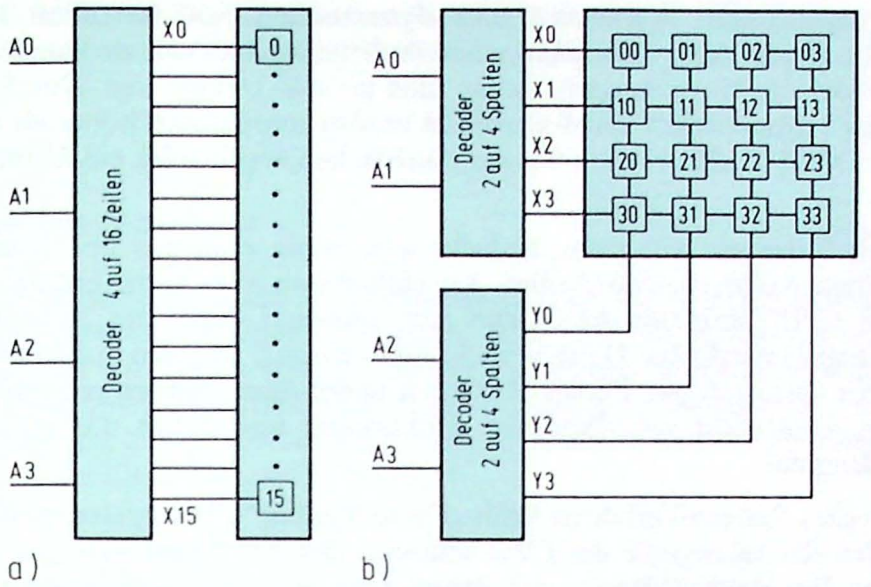


Abb. 1.3.2.2 Interne Organisation eines dynamischen Speicherchips

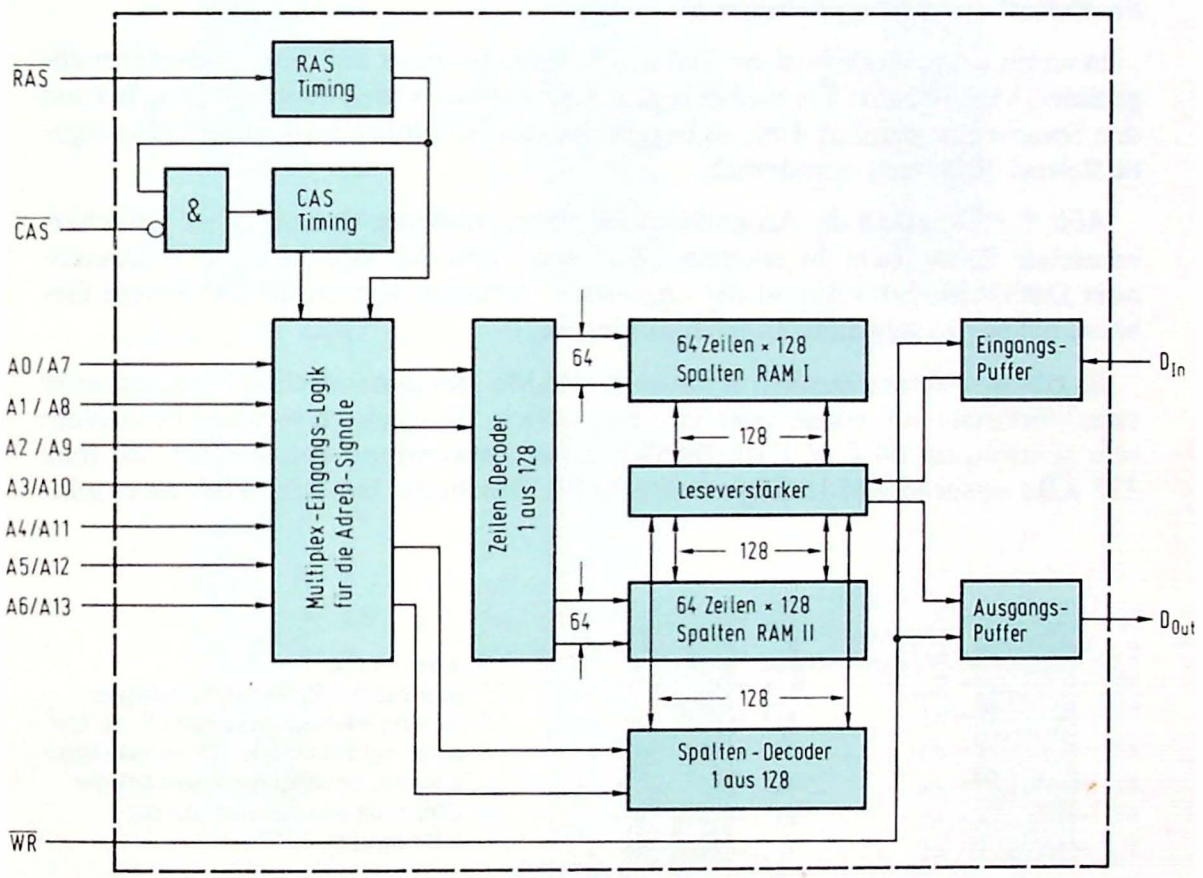


Abb. 1.3.2.3 Blockschaltbild eines 16-KBit-Speicherchips (4116)

Zum regelmäßigen Auffrischen der dynamischen Speicherzellen legt man gleichzeitig an alle Auffrischlogiken einen Auffrischpuls (v) an und aktiviert dann nacheinander die Wortleitungen, indem man sie auf 1-Pegel legt. Der Stromverbrauch eines dynamischen RAM-Bausteins ist also automatisch höher als der eines statischen MOS-RAMs, weil ja für das Auffrischen regelmäßig ein Umladestrom fließt.

Je nach Prozessor will genau bedacht sein, wann man das bei dynamischen RAMs nötige Auffrischen durchführt. Am einfachsten ist es, wenn es innerhalb des normalen CPU-Taktzyklus Abschnitte gibt, während derer mit Sicherheit kein Speicherzugriff stattfindet. Dann braucht man nämlich nur den Auffrischvorgang genau dort hineinzulegen. Dieses Verfahren nennt man „hidden refresh", weil es sozusagen unbemerkt vom Prozessor funktioniert und daher das System auch nicht verlangsamt.

Ein zweites Refresh-Verfahren heißt „Cycle Stealing" (Taktzyklen stehlen). Dabei werden die Taktsignale der CPU während des Auffrischvorganges künstlich verlängert. Der Haken dabei ist, daß das gesamte System hierdurch langsamer arbeitet. Dieser Nachteil tritt auch beim „Burst Refresh" auf: Die Schaltung generiert einen Hold-Impuls für den Prozessor (z.B. beim 8080), so daß die CPU während des Refresh einfach angehalten wird.

In vielen Computern wird ein Teil des Arbeitsspeichers auf dem Bildschirm abgebildet (Video-RAM). Da hierbei regelmäßig von der Video-Aufbereitung her auf den Speicher zugegriffen wird, ist bei geschickter Schaltungsauslegung keine eigene Refresh-Schaltung erforderlich.

Abb. 1.3.2.4 enthält die Anschlußbilder einiger üblicher dynamischer Speicherbausteine. Es ist leicht zu erkennen, daß man ohne Multiplexen von Adressen- oder Datenleitungen kaum in der Lage wäre, derartige Kapazitäten in einem Gehäuse mit nur 16 Anschlüssen unterzubringen.

Es gibt sehr unterschiedlich organisierte RAMs. Bei dynamischen Bausteinen ist eine Wortbreite von einem oder vier Bits üblich. Beispielsweise wäre es ja nicht sehr sinnvoll, ein 64 K × 1-Bit-RAM in einer Anwendung einzusetzen, wo man 128 KBit entsprechend 16 KByte mit acht Bit Wortbreite benötigt: Hier ist es gün-

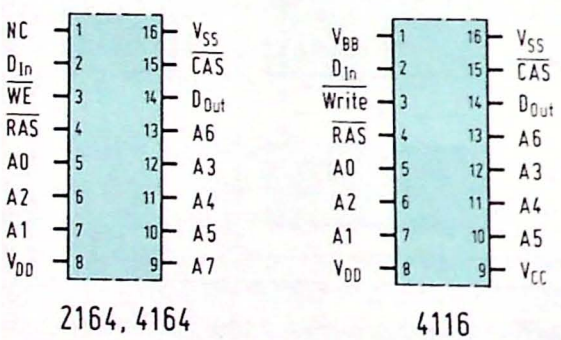


Abb. 1.3.2.4
Anschlußbilder von dynamischen 16- und 64-KBit-Speicher-Chips. Um einen acht Bit breiten Datenbus anzusteuern, benötigt man acht solcher Chips, da ihre Datenbreite nur 1 Bit beträgt

stiger, zwei 16 K × 4-Bit-RAMs zu verwenden, die zwar pro Chip die gleiche Kapazität besitzen, aber besser auf das System zugeschnitten sind.

1.3.3 Festwertspeicher

Die beiden bisher genannten Speichertypen können vom Prozessor beschrieben und gelesen werden. In jedem Computer muß aber ein gewisses Minimalprogramm resident vorhanden sein, d.h. es darf beim Ausschalten nicht verloren gehen. Sein Hauptzweck ist, überhaupt in den Computer Daten und Programme eingeben zu können. Einen Speicher, der dafür geeignet ist, nennt man Festwertspeicher oder „Read-Only Memory" (ROM).

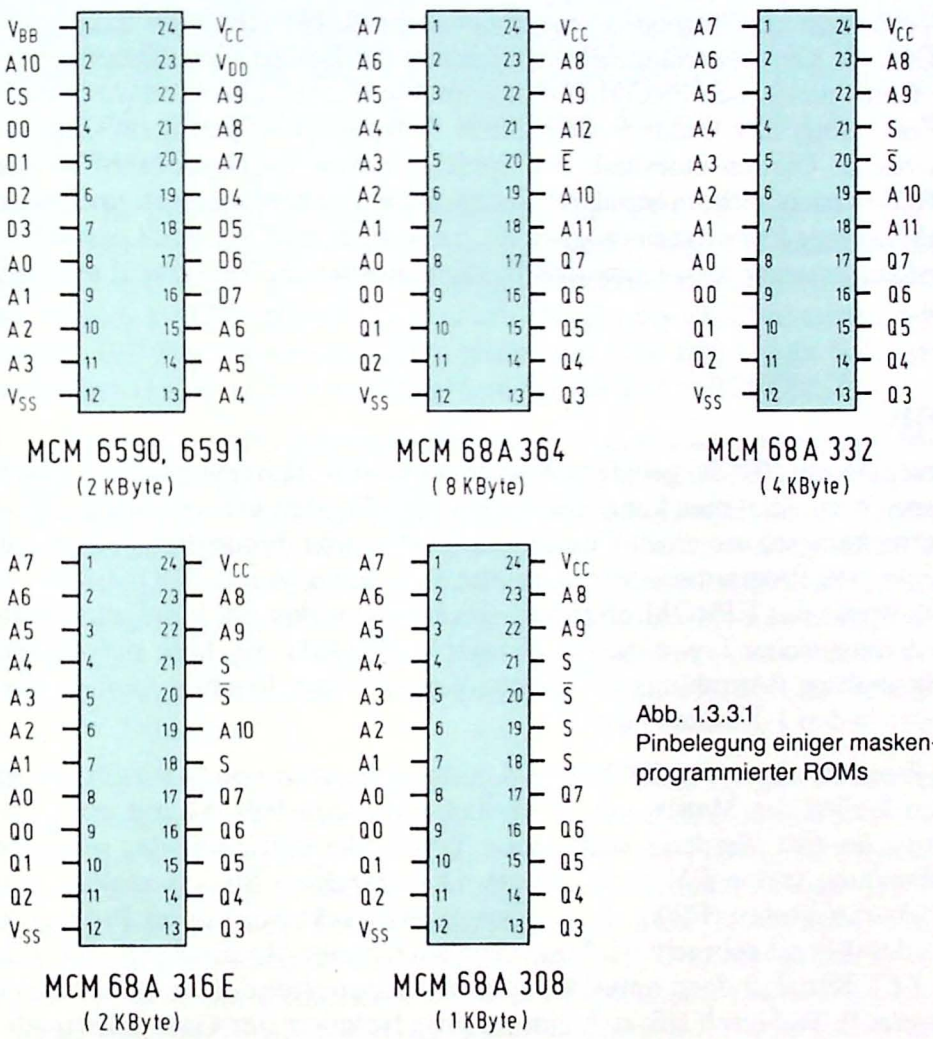


Abb. 1.3.3.1
Pinbelegung einiger maskenprogrammierter ROMs

Maskenprogrammierte ROMs

Ein „Reinkultur-ROM" wird vom Halbleiterhersteller selbst nach Kundenangaben programmiert. Das geschieht durch Anfertigen einer Maske, durch die der Silizium-Chip belichtet wird, nachdem auf ihm eine fotoempfindliche Schicht aufgebracht wurde. Bei dem nachfolgenden Ätzvorgang werden die belichteten Stellen der Speichermatrix weggeätz. Das Resultat ist das gewünschte Bitmuster. Ein solches ROM nennt man maskenprogrammiert. Es kommt natürlich nur dann in Frage, wenn einige tausend ROMs mit gleichem Inhalt hergestellt werden sollen. Maskenprogrammierte ROMs werden vorwiegend in NMOS-Technologie hergestellt, es gibt aber auch stromsparende CMOS-Typen; *Abb. 1.3.3.1* enthält eine kleine Auswahl.

PROMs

Da auch bei einem Computer, der später als Massenprodukt vermarktet werden soll, irgendwann ein Prototyp gebaut werden muß, braucht man auch eine Sorte von ROM, die der Anwender selbst individuell bei Bedarf einzeln herstellen kann. Solche Chips nennt man PROM (Programmable ROM). Das Einschreiben der Information erfolgt hier dadurch, daß durch Anlegen einer genügend hohen Spannung einzelne Dioden innerhalb der Speichermatrix durchgebrannt werden. Da sich PROMs auch leicht in bipolarer Technologie herstellen lassen und dabei auch eine sehr geringe Zugriffszeit möglich ist, verwendet man sie auch gern als Adressendecoder, um einzelnen Chips des Systems bestimmte Adressenbereiche zuzuordnen.

EPROMs

Die einmal in ein PROM geladene Information ist dort irreversibel, d.h. unlöschbar gespeichert, denn man kann durchgebrannte Dioden ja nicht mehr „heilen". In der Entwicklungsphase eines Computers ist das sehr hinderlich, denn bei jeder versuchsweisen Programmänderung müßte man einen neuen PROM-Chip opfern. Deshalb wurde das EPROM erfunden, das heute zu den am häufigsten eingesetzten Festwertspeicher-Typen gehört (Erasable PROM). Es läßt sich durch etwa zwanzigminütige Bestrahlung mit ultraviolettem Licht löschen, wobei alle Speicherzellen in den 1-Zustand kippen.

Das Programmieren von EPROMs, d.h. das Schreiben von 0-Bits in die entsprechenden Stellen der Matrix, erfolgt ähnlich wie beim PROM mit einer höheren Spannung (es gibt allerdings auch schon Typen, die dafür mit der normalen Betriebsspannung von +5 V auskommen). Die einzelnen Speicherzellen enthalten Feldeffekttransistoren (FET), deren Gate (Steuerelektrode) beim Programmieren auf 1- oder 0-Pegel gebracht wird und dort auch bleibt (floating gate). Der dazugehörige FET-Kanal ist dann entweder leitend oder sperrend, also seinerseits dauerhaft 1 oder 0. Technisch läßt sich eine so gute Isolation der Gate-Elektrode errei-

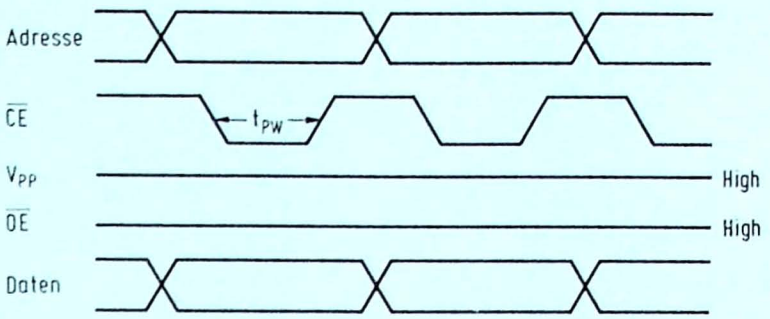


Abb. 1.3.3.2
Zeitdiagramm zum Programmieren von EPROMs. Je nach Hersteller ist es auch zulässig, bei angelegter Programmierspannung eine Zelle testweise wieder auszulesen, um die korrekte Programmierung zu kontrollieren. Manchmal (z.B. bei TI-EPROMs) muß Vpp dazu jedesmal ausgeschaltet werden

Typ	t_{PW} / ms	V_{PP} / V
2716	50	26
2732	50	26
2732 A	50	21
2764	50	21
27128	50	21
27256	50	12,5

chen, daß EPROMs die Information über einen Zeitraum von zehn bis zwanzig Jahren behalten.

Der zeitliche Ablauf des Programmierens von EPROMs geht aus *Abb. 1.3.3.2* hervor. Die genauen Zeiten und Spannungen sind typen- und herstellerabhängig. Für viele Computer gibt es EPROM-Programmierzusätze, die hardwaremäßig den Anschluß eines EPROMs über sogenannte „Nullkraftsockel" gestatten, die mit einem Hebel zur Pin-Entlastung ausgestattet sind. Die zugehörige Software kümmert sich dann um den richtigen Zeitablauf abhängig vom EPROM-Typ.

Als „intelligentes Programmierverfahren" bezeichnet man die Methode, EPROMs zunächst z.B. nur mit einem Zehntel der vom Hersteller empfohlenen Zeit zu programmieren und dann durch einen Lesevorgang zu kontrollieren, ob die Information dauerhaft gespeichert wurde. Wenn nicht, wird noch einmal z.B. ein Zehntel der insgesamt empfohlenen Zeit nachprogrammiert. Sobald ein korrektes Wiederauslesen möglich ist, erfolgt zur Sicherheit nochmals ein Nachprogrammieren z.B. mit der Zeit, die bisher insgesamt programmiert wurde. Insbesondere bei EPROMs mit hoher Kapazität ist der damit erreichbare Zeitgewinn erheblich (etwa 40...70% Einsparung).

Noch ein Wort zur Typenbezeichnung bei EPROMs. Es gibt im wesentlichen zwei Serien, nämlich die 25er und die 27er Typen. Beide unterscheiden sich geringfügig in der Pinbelegung *(Abb. 1.3.3.3)*. Die auf „25" oder „27" folgenden Ziffern geben die Kapazität in KBit (nicht KByte!) an; so besitzt z.B. der Typ 2764 eine Kapazität von 64 KBit entsprechend 8 KByte. Alle Typen außer dem 2708 kommen (bis auf den Programmiervorgang) mit einer Betriebsspannung von +5 V aus. Übrigens sind die statischen CMOS-RAM-Bausteine (siehe Kap. 1.3.1) pinkompatibel mit der 27XX-EPROM-Reihe; Unterschiede bestehen lediglich bei

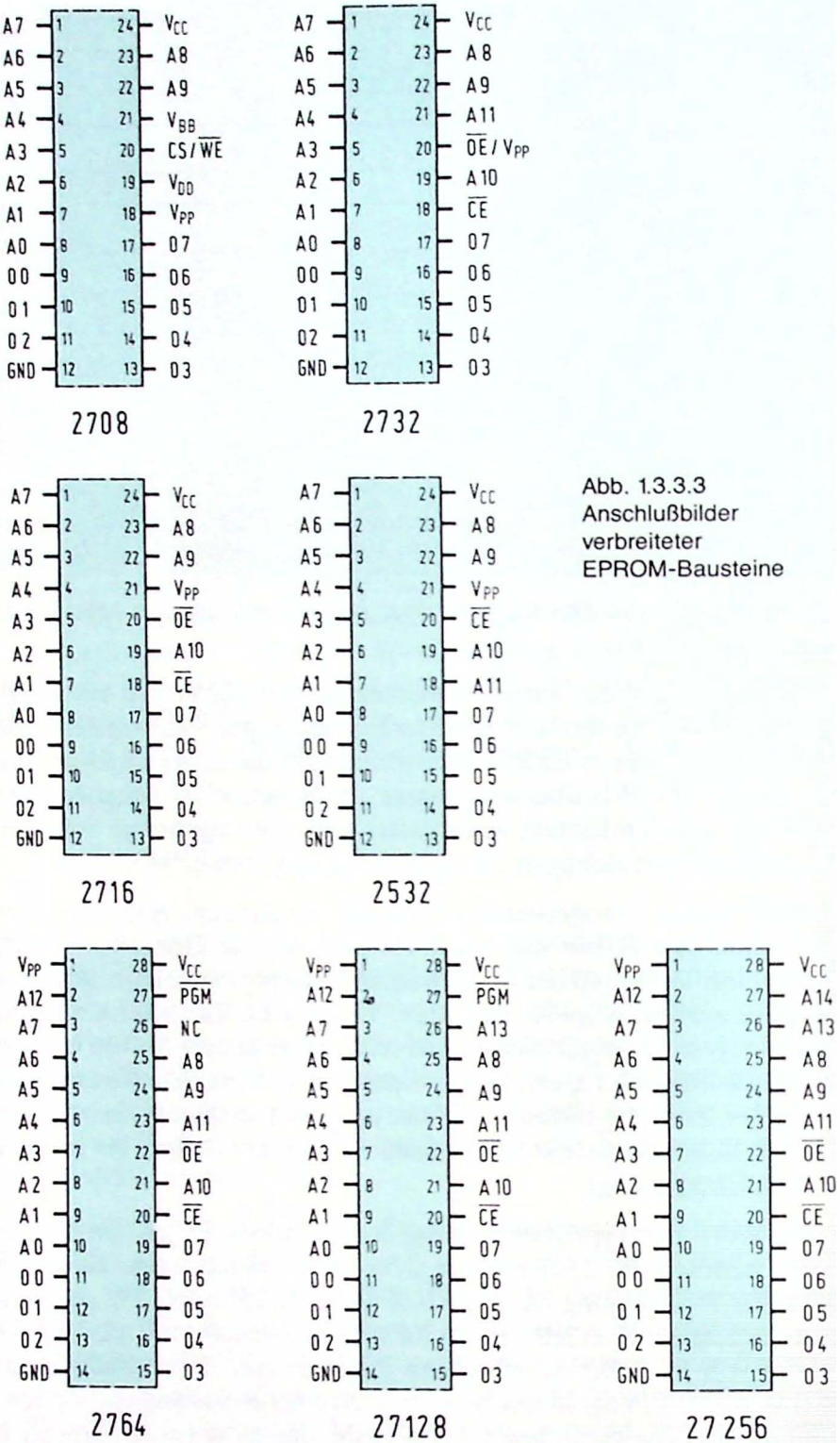


Abb. 1.3.3.3
Anschlußbilder verbreiteter EPROM-Bausteine

den Anschlüssen R/W (Read/Write beim RAM) bzw. Vpp (Programmierspannung bei EPROMs).

EEPROMs und EAROMs

Eine Weiterentwicklung der EPROMs stellen die EEPROMs dar (Electrically Erasable PROMs). Das Löschen der gespeicherten Information erfolgt bei ihnen nicht mit Ultraviolettlicht, sondern elektrisch, und auch nicht auf einmal für den gesamten Speicherinhalt, sondern wort- bzw. byteweise. EEPROMs verhalten sich also fast wie RAMs — mit zwei Unterschieden: Erstens geht die Information beim Abschalten der Versorgungsspannung nicht verloren, und zweitens ist die für einen Schreibvorgang benötigte Zeit (einige Millisekunden) um Größenordnungen länger als die Zugriffszeit beim Lesen *(Abb. 1.3.3.4)*. Löschen bedeutet hier eigentlich Überschreiben mit einem neuen Wert. Deswegen bezeichnen manche Hersteller ihre EEPROMs auch als EAROMs (Electrically Alterable ROMs). *Abb. 1.3.3.5* zeigt die Pinbelegung des EEPROMs 2816 von Intel.

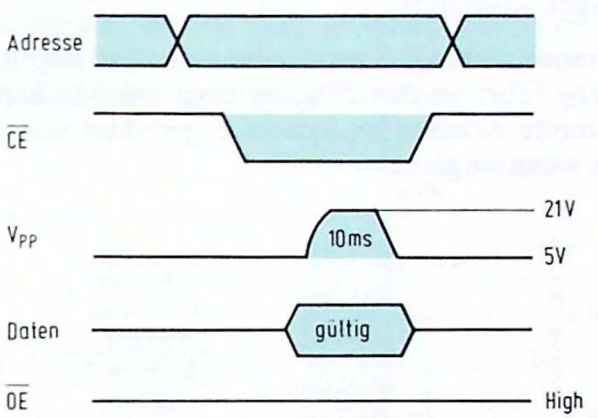


Abb. 1.3.3.4
Zeitdiagramm zum Programmieren von EEPROMs, hier am Beispiel des 2816 von Intel. Bei EEPROMs können schon programmierte Zellen einfach mit neuen Daten überschrieben werden

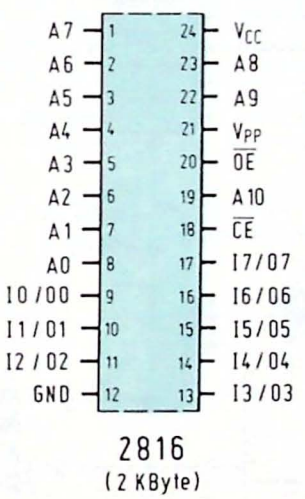


Abb. 1.3.3.5
Anschlußschema des EEPROM-Bausteins 2816 von Intel

1.4 Peripherie-Bausteine

Die Vielfalt an CPU-Typen wird von jener an Peripherie-Chips noch übertroffen. Für den Anwender recht erfreulich ist die Tatsache, daß sich bei Einhaltung elementarer Bussignal-Anforderungen Bausteine, die an sich einer bestimmten Prozessorfamilie angehören, auch mit einigen anderen CPU-Typen betreiben lassen, was die Auswahl nochmals vergrößert.

Es ist daher im Rahmen eines einzigen Buches (geschweige denn innerhalb eines Kapitels) kaum möglich, alle denkbaren Möglichkeiten anzuführen. Der folgende Überblick zeigt jedoch Typisches, das in zahlreichen Computern zu finden ist.

1.4.1 Port- und Timer-Bausteine

Die einfachste Möglichkeit, Daten von einem Computer ausgeben zu lassen, ist die Realisation sogenannter I/O-Ports (Input/Output = Ein/Ausgabe). Damit kann der Rechner z.B. über Schalttransistoren Relais, Lampen, Motoren oder Ziffernanzeigen ansteuern.

Im einfachsten Fall verwendet man dazu Flipflops *(Abb. 1.4.1.1)*. Jede Datenbus-Leitung führt an den Eingang einer solchen Kippschaltung, der gleichzeitig eine bestimmte Adresse im System zugeordnet sein muß, damit die Flipflop-Gruppe weiß, wann sie gemeint ist.

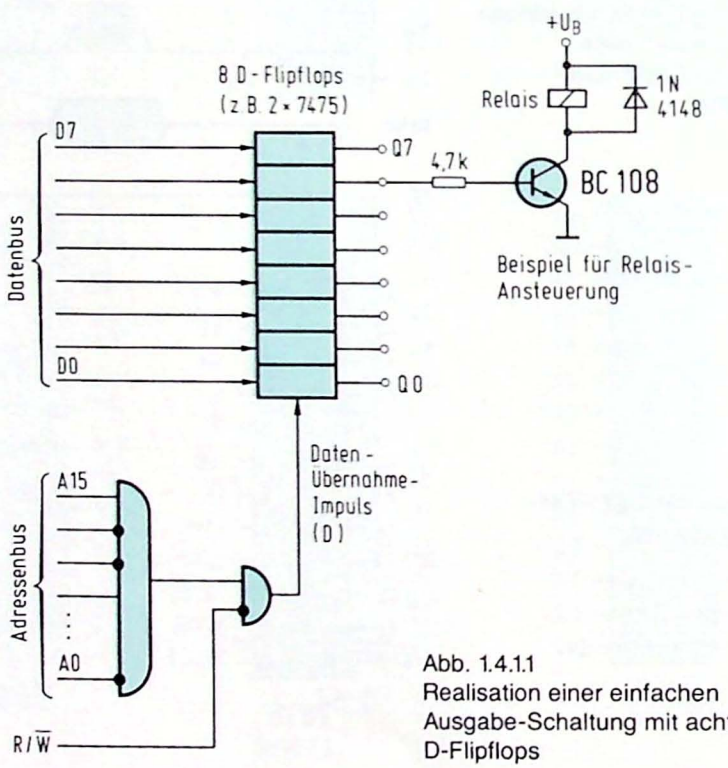


Abb. 1.4.1.1
Realisation einer einfachen Ausgabe-Schaltung mit acht D-Flipflops

Ausgabe

Die Daten vom 8-Bit-Datenbus werden nun genau dann übernommen, wenn erstens diese Adresse auf dem Adressenbus anliegt und zweitens der Prozessor über seine Schreib-/Leseleitung (R/W) der Umwelt mitteilt, daß er Daten ausgeben möchte. Die acht Bits, die dann in die acht Flipflops übernommen wurden, liegen nun an den Flipflop-Ausgängen als Spannungspegel an und können z.B. einen Schalttransistor ansteuern.

Sinnvollerweise bezeichnet man die Flipflop-Ausgänge mit den gleichen Indexziffern wie die acht Datenbusleitungen, also z.B. als Q 0...7. Angenommen, die Ausgänge Q2 und Q7 sollen auf High-Pegel (log. 1) gelegt werden, während alle anderen auf Low-Pegel liegen sollen, was muß der Prozessor dann auf dem Datenbus senden? Nichts einfacher als das: Man braucht ja nur die Ausgangszustände als Bitfolge hinschreiben, in diesem Fall 10000100 (Q7 bis Q0). Diese Dualzahl entspricht der Dezimalzahl 2↑2 + 2↑7 = 4 + 128 = 132 oder der Hexadezimalzahl 84 (vgl. Kapitel 1.1.2; diese Bitfummelei werden Sie noch oft brauchen!).

Tatsächlich kann man eine Flipflop-Schaltung, wie sie gerade beschrieben wurde, zur Ausgabe verwenden, und in mehr oder weniger aufgeblasener Form findet sie sich in den meisten Port-Bausteinen wieder. Port bedeutet hier soviel wie Schnittstelle zur Außenwelt. Bei einem 8-Bit-Datenbus ergeben sich für einen Port automatisch acht Leitungen, man spricht also von einem 8-Bit-Port.

Eingabe

Will man nun Daten oder logische Zustände von einem externen Gerät, etwa einer Tastatur, in den Mikroprozessor hineinbringen, so ist der umgekehrte Weg nötig.

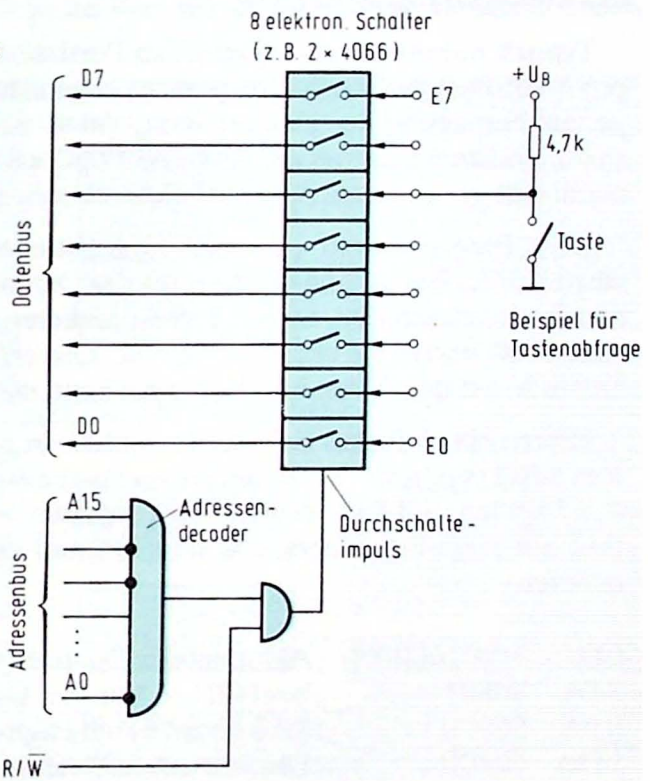


Abb. 1.4.1.2
Realisation einer einfachen Eingabe-Schaltung mit elektronischen Schaltern

Dafür braucht man nicht einmal Flipflops zur Speicherung. Vielmehr genügen elektronische Schalter, die die externen Daten wieder genau dann auf den Datenbus geben, wenn erstens eine bestimmte Adresse anliegt und zweitens die CPU über die R/W-Leitung mitteilt, daß sie nun Daten zu lesen gedenkt *(Abb. 1.4.1.2)*.

Trickreicherweise kann man für die elektronischen Schalter zur Dateneingabe sogar die gleiche Adresse vorsehen wie für die oben beschriebene Flipflop-Ausgabeschaltung. Denn ob eine Ein- oder Ausgabe stattfinden kann, ist ja durch den Zustand der R/W-Leitung eindeutig definiert. Genau so ist das in den meisten handelsüblichen Portbausteinen auch realisiert: Es gibt eine einzige, gemeinsame Adresse für Ein- und Ausgabe.

Datenrichtungs-Programmierung

Manche Bausteine gestatten es dem Programmierer, selbst festzulegen, welche ihrer Anschlüsse nun als Ein- oder als Ausgang dienen sollen. Das hat den Vorteil, daß man je nach Anwendung den Baustein optimal ausnutzen kann, ob man nun mehr Ein- oder mehr Ausgabeleitungen benötigt. Die interne Umschaltung wird in diesen Chips wiederum mit elektronischen Umschaltern vorgenommen, die von einem (nur intern verwendeten) Ausgabeport mit eigener Adresse gesteuert werden. Die Festlegung, welche Pins als Ein- bzw. Ausgänge dienen sollen, nennt man Datenrichtungs-Programmierung.

Typisch müssen einem universellen Portbaustein also wenigstens zwei Adressen pro 8-Bit-Port (bei 8-Bit-Computern) zugeordnet sein: einmal die Flipflop-Gruppe zur Festlegung der Datenrichtung (man nennt sie das „Datenrichtungs-Register"), und zum anderen die Ausgabe-Flipflops und Eingabe-Schalter, die ja, siehe oben, eine gemeinsame Adresse belegen können.

Beim Programmstart wird man zunächst einmal die gewünschte Datenrichtung einstellen, indem man an die Adresse des Datenrichtungs-Registers überall dort eine Eins hineinschreibt, wo die korrespondierende Leitung als Ausgabeleitung fungieren soll, und in die übrigen eine Null. Das erfolgt natürlich wieder in Form einer Dualzahl auf dem Datenbus. Erst dann kann man Daten ein- und ausgeben.

Hierzu ein einfaches Beispiel, formuliert in der Assembler-Sprache des Prozessors 6502 (vgl. Kap. 3.1.3). Bei einem Portbaustein, dessen Ein- und Ausgänge PA 0...7 heißen, soll PA6 als Ausgang geschaltet werden. Anschließend soll der 6502 PA6 auf High-Pegel legen und den Zustand der übrigen Leitungen in den Akku einlesen.

```
LDA  #%01000000 ;Akku mit der Dualzahl 01000000 laden (Bit 6 = 1)
STA  $0801      ;hex 0801 = Adr. des Datenrichtungs-Registers von PA
STA  $0800      ;PA6 (= Bit 6) auf High-Pegel = log. 1 legen
LDA  $0800      ;Lesen aus dem Datenregister von PA
```

Prinzipiell funktioniert das bei anderen Prozessortypen ganz ähnlich. Allerdings gibt es bei manchen CPUs (8080, 8085, Z80 u.a.) einen eigenen IORQ-Anschluß (I/O Request). Er dient zur Unterscheidung zwischen dem normalen Speicher- und einem zweiten, getrennten I/O-Adressenbereich. In der Assembler-Sprache dieser Prozessoren gibt es deshalb auch bestimmte Befehle, die speziell für Ein-/Ausgabeoperationen vorgesehen sind (IN, OUT) und die die IORQ-Leitung umschalten. Das hat einerseits den Vorteil, daß der (speziell bei 8-Bit-CPUs mit nur 64 KByte recht kostbare) Adressenraum nicht zum Teil für I/O-Aufgaben „verschwendet" werden muß, aber andererseits auch den Nachteil, daß sich nicht alle Prozessorbefehle auch auf I/O-Bausteine anwenden lassen, wie dies beim „memory-mapped"-Konzept von 6502, 6800 und anderen CPUs ohne weiteres möglich ist.

Z80-PIO

Die Bausteine 8255 und Z80-PIO entsprechen weitgehend dem eben geschilderten I/O-Konzept. *Abb. 1.4.1.3* zeigt das „Innenleben" der Z80-PIO (Parallel I/O; im Sprachgebrauch hat sich „die" PIO eingebürgert).

Die Z80-PIO enthält zwei Ports (A und B). Zu jedem Port gehören zwei weitere Steuerleitungen (Strobe und Ready). Eine besondere Eigenschaft ist auch die Fähigkeit, bei bestimmten, programmierbaren Datenkonstellationen einen Interrupt auszulösen. Der Baustein kennt für jeden Port vier Betriebsarten, die durch Laden

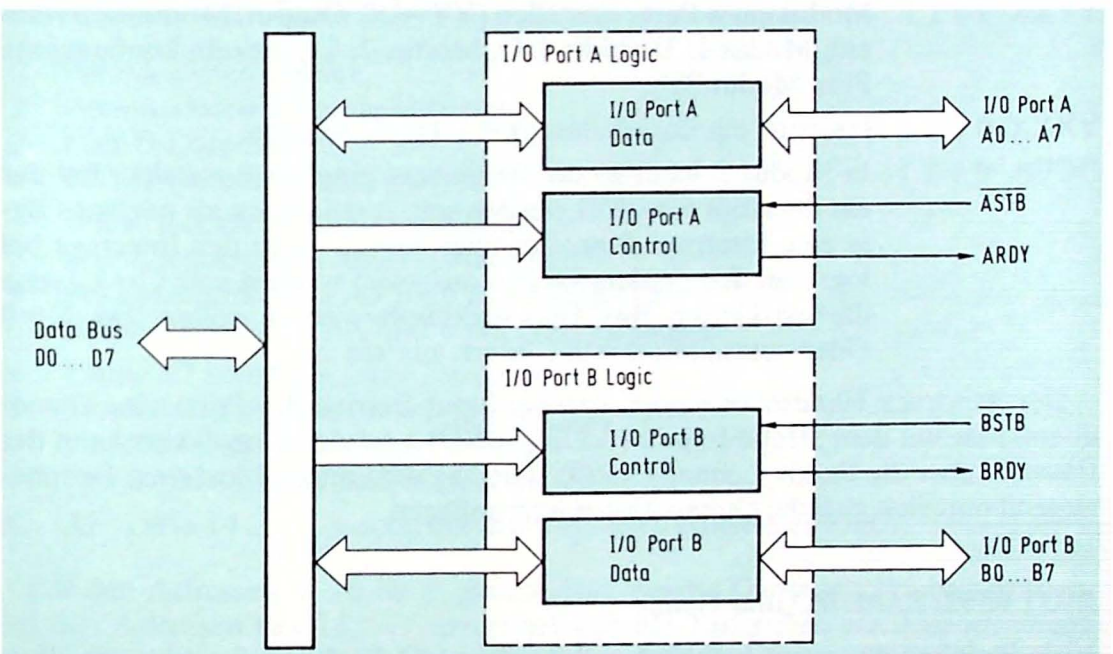


Abb. 1.4.1.3 Interner Aufbau des Bausteins Z80-PIO mit zwei 8-Bit-Ports

eines Steuerworts in ein Kontrollregister eingestellt werden; der 8255 dagegen nur drei:

Betriebsart	Z80-PIO	8255
Output mit Handshake	Modus 0	Modus 1
Input mit Handshake	Modus 1	Modus 1
Bidir. I/O m. Handsh.	Modus 2	Modus 2
Einf. Inp. oder Outp.	Modus 3	Modus 0
Pins einzeln konfig.	Modus 3	nicht vorh.

Innerhalb des Adressenbereichs, der durch ein Low-Signal an $\overline{CE}$- ausgewählt wird, gibt es bei der Z80-PIO vier Adressen (relativ zur Grundadresse), wenn man B/$\overline{A}$- mit A1 und C/$\overline{D}$- mit A0 verbindet:

0 Port-A-Datenregister
1 Port-A-Steuerregister
2 Port-B-Datenregister
3 Port-B-Steuerregister

Das Kommandoformat wird durch die niederwertigen Bits des Bytes festgelegt, das in ein Steuerregister geschrieben wird:

7654 3210	Bit-Nr.
XXXX XXX0	In Modus 2 kann so ein Interrupt-Vektor spezifiziert werden. Die PIO antwortet dann mit XXXXXXX0 auf ein Interrupt-Acknowledge-Signal der Z80-CPU.
YYXX 1111	Modus eines Ports einstellen (YY = 00: Output, Modus 0; 01: Input, Modus 1; 10: bidir. I/O, Modus 2; 11: einzeln konfigurierte Pins, Modus 3)
YXXX 0011	Interrupt ein/ausschalten: Y = 1: ein; Y = 0: aus
DCBA 0111	In Modus 3 kann so die Bedingung eingestellt werden, bei der ein Interrupt ausgelöst werden soll. A = 1, wenn als nächstes Byte eine Interrupt-Bitmaske folgt; B = 1, wenn der Interrupt bei log. 0 an den maskierten Bits ausgelöst werden soll; C = 1, wenn die maskierten Bits Und-verknüpft werden sollen, bei C = 0 Oder-verknüpft; D = 0/1 Interr. aus/ein

Der Ausdruck Handshake besagt, daß bei Input-Betrieb des Ports eine Datenübernahme mit dem Strobe-Impuls ($\overline{ASTB}$-, $\overline{BSTB}$-) erfolgt. Umgekehrt kann der Baustein über die Ready-Leitung (ARDY, BRDY) dem angeschlossenen Peripheriegerät mitteilen, daß die Output-Daten jetzt anliegen.

RIOT 6532: RAM, I/O und Timer

Viele Peripheriebausteine enthalten außer reinen I/O-Funktionen auch noch einen Timer. Der zur 65XX-Familie gehörende 6532 besitzt darüber hinaus auch noch

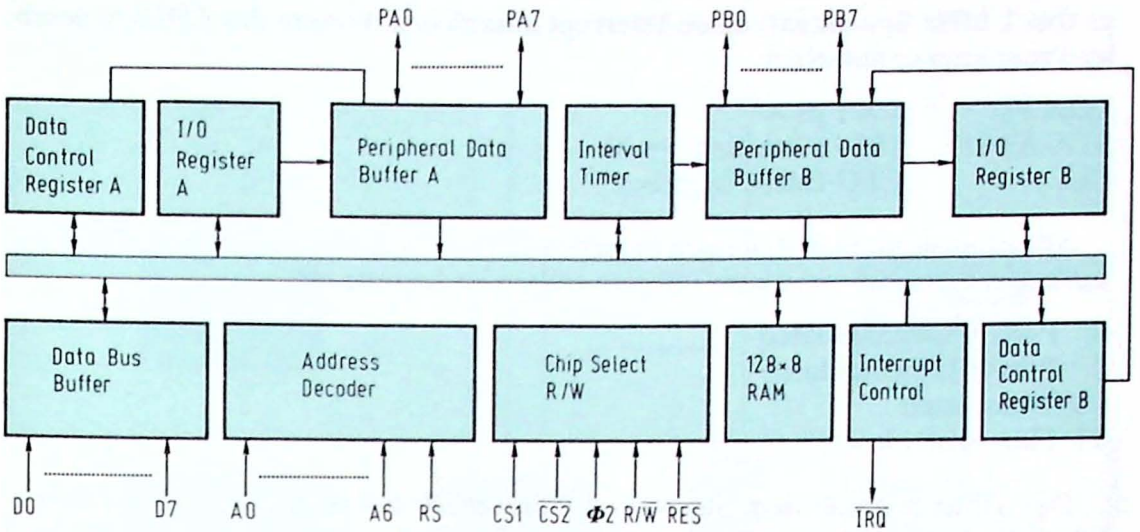


Abb. 1.4.1.4 Blockschaltbild des RIOT-Bausteins 6532 mit Timer, zwei 8-Bit-Ports und 128 Byte RAM

128 Byte statisches RAM *(Abb. 1.4.1.4)*. Er eignet sich dadurch speziell für den Aufbau kleiner Systeme wie z.B. Einplatinencomputer, weil er die Anzahl der nötigen Chips erheblich reduziert. (Einen Timer-Baustein gibt es übrigens auch in der Z80-Familie, er heißt Z80-CTC).

Die relativ zur Grundadresse gerechneten Adressen zum Schreiben des 6532 lauten ($\overline{RS}$- = High, R/$\overline{W}$- = Low):

0 Port-A-Datenregister
1 Port-A-Datenrichtungs-Register
2 Port-B-Datenregister
3 Port-B-Datenrichtungs-Register
4 Wie 6, jedoch auch Interrupt auslösen
5 Wie 7, jedoch auch Interrupt auslösen
6 Bei pos. Flanke an PA7 Bit 6 im Flagreg. setzen
7 Bei neg. Flanke an PA7 Bit 6 mit Flagreg. setzen
14 Timer 1 Taktzyklus, ohne Interrupt
15 Timer 8 Taktzyklen, ohne Interrupt
16 Timer 64 Taktzyklen, ohne Interrupt
17 Timer 1024 Taktzyklen, ohne Interrupt
1C...1F Wie 14...17, jedoch mit Interrupt nach Ablauf der Zeit

Bei den Adressen 4...6 ist es gleichgültig, welche Daten geschrieben werden. Bei den Adressen hex 14...1F errechnet sich die Timer-Zeit aus dem jeweiligen Multiplikator (1, 8, 64, 1024), dem CPU-Taktzyklus (1 µs bei 1 MHz) und den Daten auf dem Datenbus. Will man beispielsweise den Timer veranlassen, nach 4096

µs (bei 1 MHz Systemtakt) einen Interrupt auszulösen, könnte das 6502-Assembler-Programm so aussehen:

```
LDA #4        ;4 × 1 µs ×
STA A+$1F     ;1024 (Grundadr. = A)
CLI           ;CPU-Interr. freigeben
```

Alle oben genannten Adressen beziehen sich auf Schreiboperationen. Beim Lesen kommt den Adressen zum Teil eine andere Bedeutung zu:

0 Port-A-Eingangsdaten
2 Port-B-Eingangsdaten
16 Timer lesen
17 Flagregister lesen

Der Timer zählt ab dem Startwert kontinuierlich mit dem eingestellten Teilerfaktor (1, 8, 64, 1024) nach unten. Sobald er bei Null angekommen ist, wird Bit 7 des Flagregisters auf 1 gesetzt. Will man also beispielsweise in einem Programm rund 8 ms warten, so kann das wie folgt geschehen:

```
      LDA #8          ;8 × 1024 µs
      STA RIOT+$17    ;Teilerfaktor 1024
LOOP  BIT RIOT+$17    ;Flagregister lesen
      BPL LOOP        ;Schleife, bis Zeit abgelaufen
```

Die 128 Byte RAM werden selektiert, wenn $\overline{\text{RS}}$- (RAM Select) auf Low liegt.

VIA 6522: Versatile Interface Adapter

Ein intern noch komplexerer Baustein, obwohl kein RAM auf dem Chip ist, ist die VIA 6522. Sie besitzt zwei 16-Bit-Timer, die grundsätzlich anders organisiert sind als der Timer im 6532, sowie ein 8-Bit-Schieberegister und zwei I/O-Ports PA und PB *(Abb. 1.4.1.5)*. Die Adressenlage der insgesamt 16 Register sieht so aus:

0 Port-B-Datenregister
1 Port-A-Datenregister mit Handshake
2 Port-B-Datenrichtungs-Register
3 Port-A-Datenrichtungs-Register

	Funktion beim Schreiben:	*Funktion beim Lesen:*
4	T1L-L schreiben	T1C-L lesen, Interruptflag löschen
5	T1L-H schreiben, T1L-L nach T1C-L	T1C-H lesen
6	T1L-L schreiben	T1L-L lesen
7	T1L-H schreiben, Interruptflag löschen	T1L-H lesen
8	T2L-L schreiben	T2C-L lesen, Interruptflag löschen

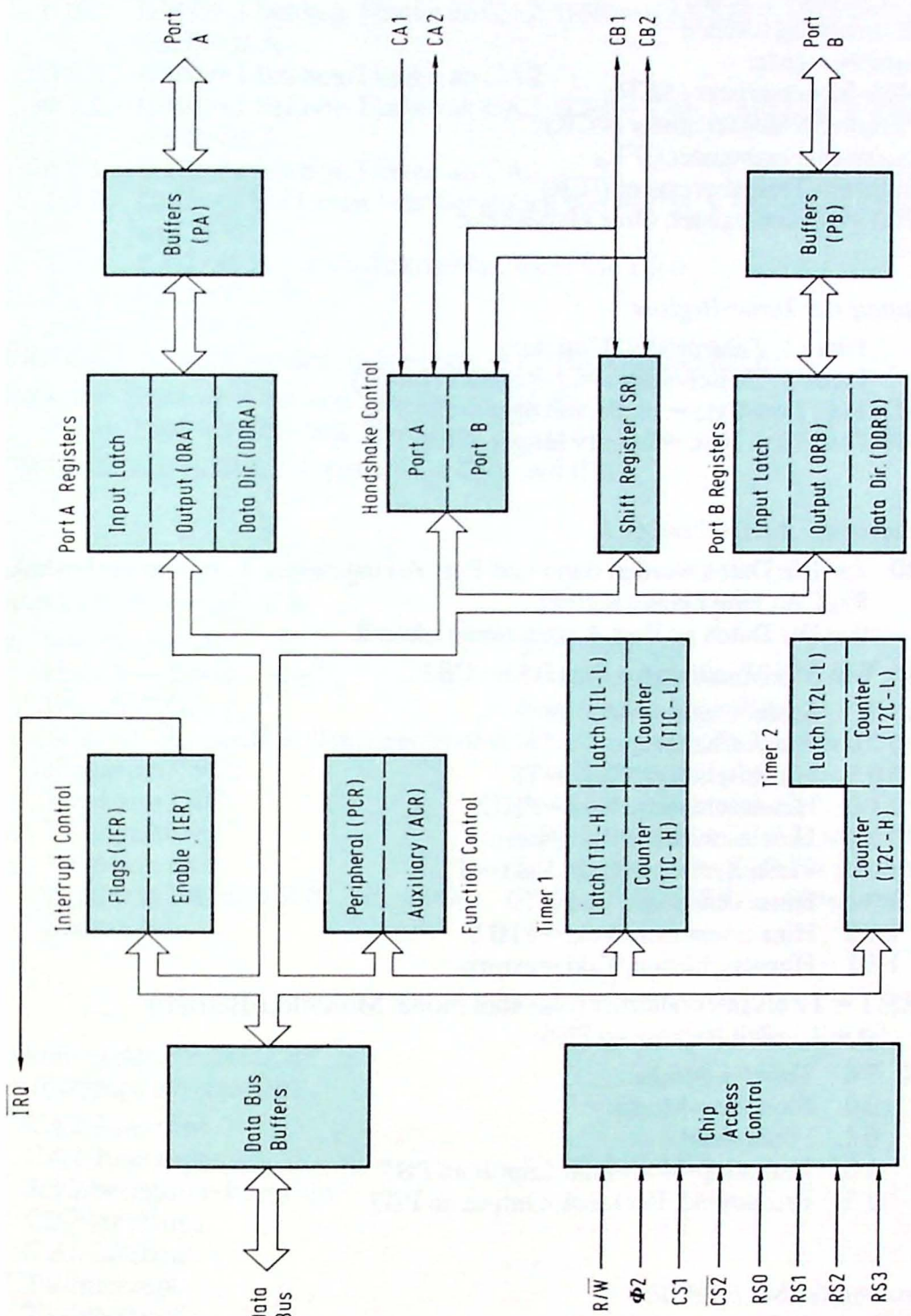


Abb. 1.4.1.5 Aufbau des VIA-Bausteins 6522 (zwei 8-Bit-Ports, zwei Timer, ein 8-Bit-Schieberegister)

9 T2C-H schreiben, T2C-H lesen
T2L-L nach T2C-L,
Interruptflag löschen
A Schieberegister
B Hilfs-Steuerregister (ACR)
C Peripherie-Steuerregister (PCR)
D Interrupt-Flagregister (IFR)
E Interrupt-Freigaberegister (IER)
F Port-A-Datenregister, ohne Handshake

Bedeutung der Timer-Register:

T1C Timer 1, Zählregister (Counter)
T1L Timer 1, Zwischenspeicher-Register (Latch)
T1C-L T1C, Low Byte = niederwertiger 8-Bit-Teil
T1C-H T1C, High Byte = höherwertiger 8-Bit-Teil

Bedeutung der Bits 0—7 im ACR:

ACR0: 1 = Die Daten werden dann von Port A eingelesen, wenn am Handshake-Pin CA1 eine Flanke auftritt.
0 = Die Daten an Port A sind immer aktuell.

ACR1: Wie ACR0, jedoch für Port B und CB1.

ACR4 3 2 Schieberegister-Modus:
0 0 0 Ausgeschaltet
0 0 1 Hineinschieben, Takt = T2
0 1 0 Hineinschieben, Takt = PHI2
0 1 1 Hineinschieben, Takt extern
1 0 0 Freilaufender Output, Takt = T2
1 0 1 Hinausschieben, Takt = T2
1 1 0 Hinausschieben, Takt = PHI2
1 1 1 Hinausschieben, Takt = extern

ACR5: 1 = T2 als Intervalltimer (one-shot mode, Monoflop-Betrieb)
0 = T2 zählt Impulse an PB6

ACR 7 6 Timer-1-Modus:
0 0 Monoflop-Modus
0 1 Freilaufend
1 0 Monoflop-Modus mit Impuls an PB7
1 1 Freilaufend, Reckteck-Output an PB7

Bedeutung der Bits im PCR:

PCR0: 1 = IFR1 wird bei negativer Flanke an CA1 gesetzt.
0 = IFR1 wird bei positiver Flanke an CA1 gesetzt.

PCR3 2 1 CA2-Betriebsart:

- 0 0 0 IFR0 = 1 bei neg. Flanke an CA2, IFR0 = 0 bei Lesen oder Schreiben des Ports A
- 0 0 1 IFR0 = 1 bei neg. Flanke an CA2
- 0 1 0 IFR0 = 1 bei pos. Flanke an CA2, IFR0 = 0 bei Lesen oder Schreiben des Ports A
- 0 1 1 IFR0 = 1 bei pos. Flanke an CA2
- 1 0 0 CA2 = 1 bei Lesen oder Schreiben von Port A, CA2 = 0 bei Flanke an CA1
- 1 0 1 CA2 = 1 für einen Taktzyklus, sonst wie 1 0 0
- 1 1 0 CA2 = 0
- 1 1 1 CA2 = 1

(IFR0 kann gelöscht werden, indem man ein 1-Bit hineinschreibt!)

PCR4: 0 = IFR4 wird bei neg. Flanke an CB1 gesetzt.
1 = IFR4 wird bei neg. Flanke an CB1 gesetzt.

PCR7, 6, 5: Wie PCR3, 2, 1, jedoch für CB2 und IFR3.

Bedeutung der Bits im IFR:

Bit	Gesetzt durch:	Rücksetzen:
0	Flanke an CA2	Port A lesen/schreiben
1	Flanke an CA1	Port A lesen/schreiben
2	Schiebereg. wurde achtmal geschoben	Schiebereg. lesen/schreiben
3	Flanke an CB2	Port B lesen/schreiben
4	Flanke an CB1	Port B lesen/schreiben
5	T2 abgelaufen	T2C-L lesen oder T2C-H schreiben
6	T1 abgelaufen	T1C-L lesen oder T1C-H schreiben
7	Irgendein Bit im IFR (0...6) wurde gesetzt	IFR0...6 oder IER0...6 löschen

Bedeutung der Bits im IER:
(1 = Interrupt einschalten)

- 0 CA2-Interrupt
- 1 CA1-Interrupt
- 2 Schieberegister-Interrupt
- 3 CB2-Interrupt
- 4 CB1-Interrupt
- 5 T2-Interrupt
- 6 T1-Interrupt
- 7 0 = Mit 1-Bits entsprechende IER-Bits löschen
  1 = Mit 1-Bits entsprechende IER-Bits setzen

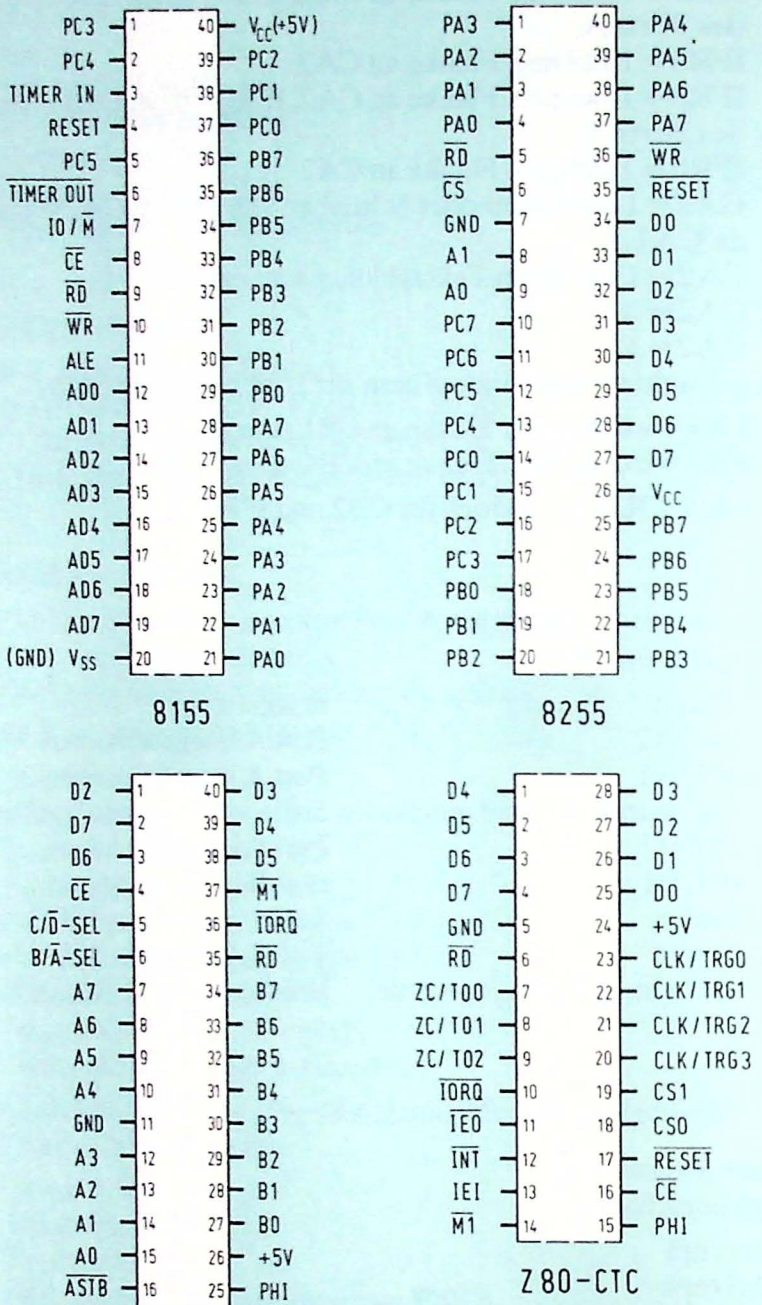
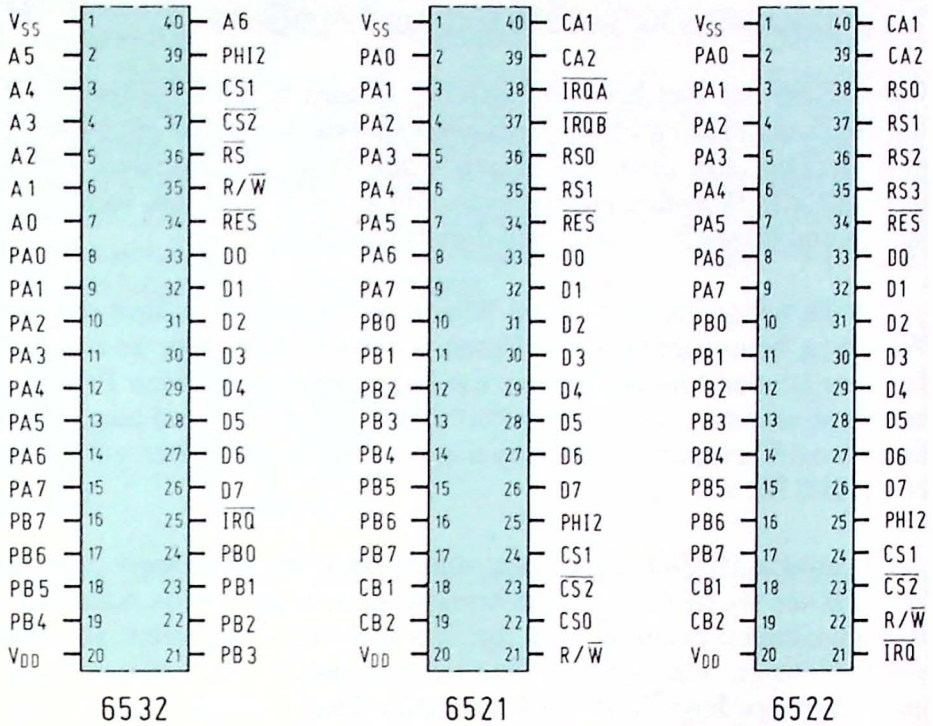


Abb. 1.4.1.6
Anschlußbilder der wichtigsten Peripherie-Bausteine

zu Abb. 1.4.1.6



Pinbelegungen einiger Port- und Timer-Bausteine

Abb. 1.4.1.6 zeigt die Anschlußbelegung einiger handelsüblicher Port- und Timer-Bausteine. Die dabei verwendeten Abkürzungen sind:

PA, PB, PC	Port-I/O-Leitungen
A, B	Port-I/O-Leitungen bei Z80-PIO
A, D, AD	Adressen- und Datenbus
IEI	Interrupt-Freigabe-Eingang
IEO, IRQ	Interrupt-Ausgang
CS	Chip-Select (vom Adressendecoder)
CE	Chip Enable (ähnl. CS)
CA, CB, ASTB, BSTB, ARDY, BRDY	Handshake-Steuerleitungen
RS	Register-Select-Eingang
R/W, WR, RD	Read/Write-Eingang
ZC/TO	Zero Count/Time Out (Zeit abgelaufen)
TIMER IN	Timer-Eingang
TIMER OUT	Timer-Ausgang
CLK/TRG	Takt- und Trigger-Anschluß
RES, RESET	Hardware-Reset-Eingang
PHI, M1	Taktsignale von der CPU
IORQ, IO/M	Umschaltung von Speicher- auf I/O-Adr.-Bereich
VCC, VDD	+5 V
VSS	Masse (Ground)

1.4.2 Bausteine für serielle Ein- und Ausgabe

Wie wir bei der Beschreibung serieller Schnittstellen in Kapitel 1.7 noch sehen werden, kann man erheblich Leitungen sparen, wenn die einzelnen Bits zwischen zwei Geräten nicht parallel, sondern seriell übertragen werden. Das bedeutet, daß nicht alle acht Bits eines Bytes gleichzeitig auf acht Drähten, sondern nacheinander auf ein und demselben Draht transferiert werden.

Um die nötige Parallel/Seriell-Wandlung beim Senden und die Seriell/Parallel-Wandlung beim Empfang von Daten durchzuführen, gibt es zwei Möglichkeiten. Die eine ist, den Mikrocomputer mit einem entsprechenden Programm zu veranlassen, an einem beliebigen I/O-Port Daten seriell ein- und auszugeben. Das funktioniert natürlich nur bei relativ geringen Geschwindigkeiten zufriedenstellend (bis etwa 1200 Bit/s).

Die zweite Möglichkeit besteht darin, spezielle Hardware-Bausteine einzusetzen, z.B. die ACIA 6551 (Asynchronous Communications Adapter). Asynchrone Kommunikation bedeutet, daß die Bits nicht in konstantem Rhythmus gesendet werden müssen. Vielmehr werden Gruppen von je z.B. 8 Bits gesendet. Daß solch eine Bitgruppe folgt, wird durch ein besonderes Startbit (log. 0) angekündigt, das ihr vorausgeht. Nach dem Ende der Bitfolge nimmt die Übertragungsleitung für mindestens eine Bitlänge (Stopbit) den Ruhezustand log. 1 ein *(Abb. 1.4.2.1)*.

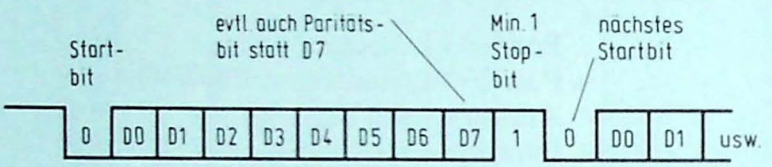


Abb. 1.4.2.1 Zeitdiagramm einer seriellen, asynchronen Datenübermittlung z.B. über V.24-Schnittstellen

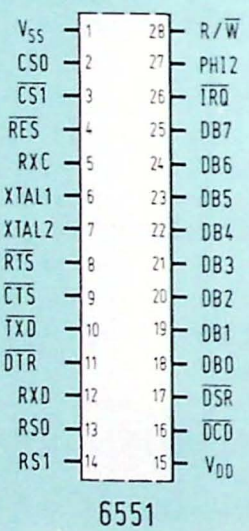


Abb. 1.4.2.2
Anschlußschema des Bausteins 6551. Er besitzt einen eingebauten Baudraten-Generator und erzeugt bzw. verwertet alle nötigen V.24-Steuersignale

Die Pinbelegung des ACIA-Bausteins 6551 geht aus *Abb. 1.4.2.2* hervor. Intern besteht er im wesentlichen aus zwei Schieberegistern, einem zum Senden, einem zum gleichzeitigen Empfangen von Daten (Duplexbetrieb). Entsprechend der V.24-Schnittstellen-Norm werden auch zusätzliche Steuerleitungen zur Verfügung gestellt. Die Verbindung mit der CPU des Systems erfolgt direkt über Adressen- und Datenbus. Über einen Adressendecoder werden die Chip-Select-Signale CS0 und $\overline{CS1}$- angesteuert. Die Register-Select-Leitungen RS0 und RS1 verbindet man mit den Adressenbusleitungen A0 und A1. Bezogen auf die jeweilige Grundadresse ergeben sich dann folgende Registeradressen:

Adr.	Schreiben	Lesen
0	Sendedaten	Empfangsdaten
1	Reset-Befehl	Statusregister
2	Kommandoreg.	Kommandoreg.
3	Steuerregister	Steuerregister

Die Daten, die bei einem Reset-Befehl in Register 1 geschrieben werden, spielen keine Rolle.

Die Bits im Kommandoregister haben folgende Bedeutung:

Bit 7 6 5	Wirkung:
X X 0	Kein Parity-Bit
0 0 1	Senden und Empfangen mit ungerader Parität
0 1 1	Senden und Empfangen mit gerader Parität
1 0 1	Senden: Parity = 1; Empfang: Parity nicht ausgewertet
1 1 1	Senden: Parity = 0; Empfang: Parity nicht ausgewertet

Bit 4: 0: Kein Echo
1: Echo (empf. Zeichen werden wieder gesendet; Bits 2 und 3 müssen dabei 0 sein)

Bit 3 2	Sender-Interr.	RTS-Pegel	Sender
0 0	nein	high	aus
0 1	ja	low	ein
1 0	nein	low	ein
1 1	nein	low	BRK senden

Bit 1: 0: Interrupt wird durch Bit 3 im Statusreg. ausgelöst
1: Kein Interrupt

Bit 0: 0: Empfangsteil und Interrupts abschalten, $\overline{DTR}$- high
1: Empfangsteil ein, Interrupts zulassen, $\overline{DTR}$- low

(Alle Bits werden durch einen Hardware-Reset auf Null gesetzt, bei einem Software-Reset aber keines.)

Bits im Steuerregister

Bit 7 0: 1 Stopbit
1: a) bei 8 Daten- und 1 Paritybit: 1 Stopbit
b) bei 5 Daten- und kein Paritybit: 1,5 Stopbits
c) sonst 2 Stopbits

Bit 6 5 Datenbits:
0 0 8
0 1 7
1 0 6
1 1 5

Bit 4 0: Externer Empfangstakt
1: Eingebauter Taktgenerator als Empfangstakt

Bit 3 2 1 0 Baudrate (Bit/s):
0 0 0 0 1/16mal externer Takt
0 0 0 1 50 Bd
0 0 1 0 75 Bd
0 0 1 1 109,92 Bd
0 1 0 0 134,58 Bd
0 1 0 1 150 Bd
0 1 1 0 300 Bd
0 1 1 1 600 Bd
1 0 0 0 1200 Bd
1 0 0 1 1800 Bd
1 0 1 0 2400 Bd
1 0 1 1 3600 Bd
1 1 0 0 4800 Bd
1 1 0 1 7200 Bd
1 1 1 0 9600 Bd
1 1 1 1 19200 Bd

(Die Bits 0...3 werden bei einem Software-Reset gelöscht, bei einem Hardware-Reset alle.)

Bits im Statusregister:

Bit	Bedeutung, wenn = 1:	Rücksetzen:
7	Paritätsfehler	Automatisch
6	Frame-Error	Automatisch
5	Overrun-Error	Automatisch
4	Wort empfangen	Lesen des Empfangsdaten-Registers
3	Sendedaten gesendet	Schreiben in Sendedaten-Register
2	DCD-Zustand	—
1	DSR-Zustand	—
0	Interrupt	Lesen des Statusregisters

(Ein Software-Reset löscht nur Bit 2, ein Hardware-Reset löscht alle Bits.)

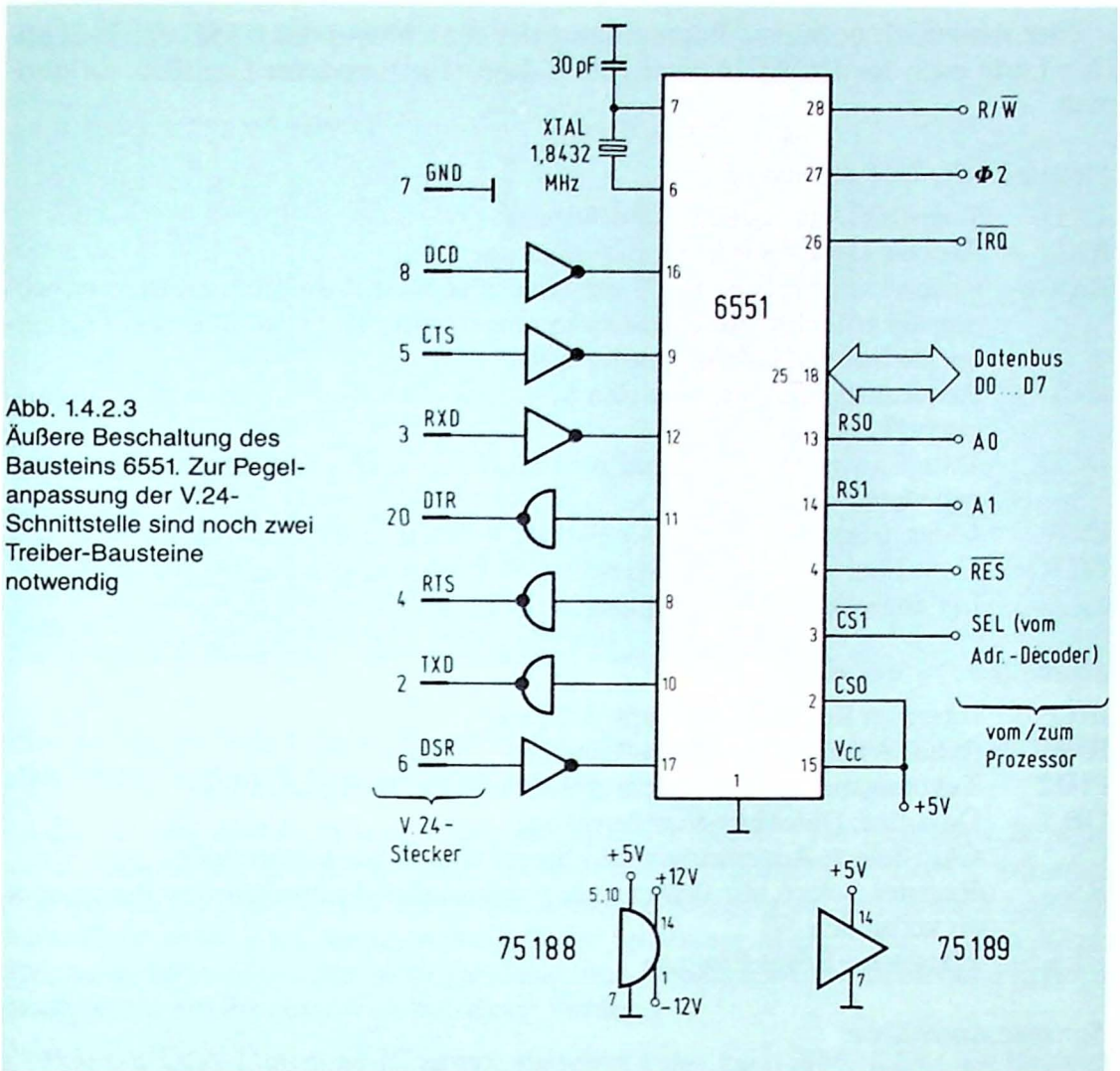


Abb. 1.4.2.3
Äußere Beschaltung des Bausteins 6551. Zur Pegelanpassung der V.24-Schnittstelle sind noch zwei Treiber-Bausteine notwendig

Der Baustein 6551 wird normalerweise mit einem 1,8432-MHz-Quarz zwischen den XTAL-Anschlüssen betrieben. Zusätzlich sollte man einen 30-pF-Kondensator von Pin 7 nach Masse schalten. Nähere Angaben über die realisierte V.24- bzw. RS-232-Schnittstelle finden sich in Kapitel 1.7. *Abb. 1.4.2.3* zeigt die Schaltung einer realisierten V.24-Schnittstelle mit dem 6551.

Neben dem 6551 (als 65C51 oder 65SC51 auch in stromsparender CMOS-Technologie erhältlich) gibt es auch in anderen Prozessor-Familien Bausteine mit ähnlichen Eigenschaften, in der Z80-Familie beispielsweise Z80-DART (wie der 6551 für asynchrone Übertragung mit Start- und Stopbits) und Z80-SIO (für asynchrone und synchrone Übertragung). Darüber hinaus gibt es sogenannte UART-Bausteine (Universal Asynchronous Receiver/Transmitter), die allerdings nicht direkt an den Adressen- und Datenbus angeschlossen, sondern über einen I/O-Port parallel angesteuert werden.

Hier schließlich noch eine Beschreibung der Anschlüsse des 6551, die in ähnlicher Form auch bei den ACIA- und UART-Bausteinen anderer Familien vorkommen:

Signale an der V.24-Schnittstelle:

TXD	Transmit Data; serielle Sendeleitung
RXD	Receive Data; serielle Empfangsleitung
RXC	Receive Clock; beim 6551 ein bidirektionaler Anschluß, an dem entweder die 16fache Baudrate extern eingespeist oder vom internen Oszillator nach außen geführt werden kann
RTS	Request to Send; schaltet den Sendeteil eines angeschlossenen Modems ein (vgl. Kap. 2.2)
DCD	Data Carrier Detect; Signal vom Modem, ob eine Gegenstation empfangen wird
CTS	Clear to Send; Meldung der Betriebsbereitschaft vom Modem
DTR	Data Terminal Ready; Signal zum Modem, daß der ACIA-Baustein bzw. der angeschlossene Computer bereit ist

Signale am Prozessorbus:

IRQ	Interrupt Request; Interrupt-Ausgang
R/W	Read/Write; Schreib/Lese-Eingang
PHI2	Takteingang für Bus-Synchronisation (nicht für Baudrate!)
DB	Data Bus; Datenbus-Ein/Ausgänge
CS	Chip Select; Adressenauswahl-Signal vom Adressendecoder
RS	Register Select; mit den entspr. Adressenbus-Leitungen des Prozessors zu verbinden
RES	Hardware-Reset-Eingang

Sonstige Anschlüsse:

XTAL	Anschlüsse für den Quarz, aus dem durch Frequenzteilung die gewünschte Baudrate abgeleitet wird
VSS	Masse (0 V)
VDD	+5 V

1.4.3 D/A- und A/D-Umsetzer

D/A-Umsetzer

Mikrocomputer arbeiten intern rein binär, also mit nur zwei Spannungspegeln. Zwischenwerte sind unerwünscht. Nun gibt es aber viele Anwendungen, z.B. in der Steuerungs- und Regelungstechnik, bei denen es nötig ist, daß der Computer eine stetig veränderliche Größe programmgesteuert erzeugt. Damit kann man z.B.

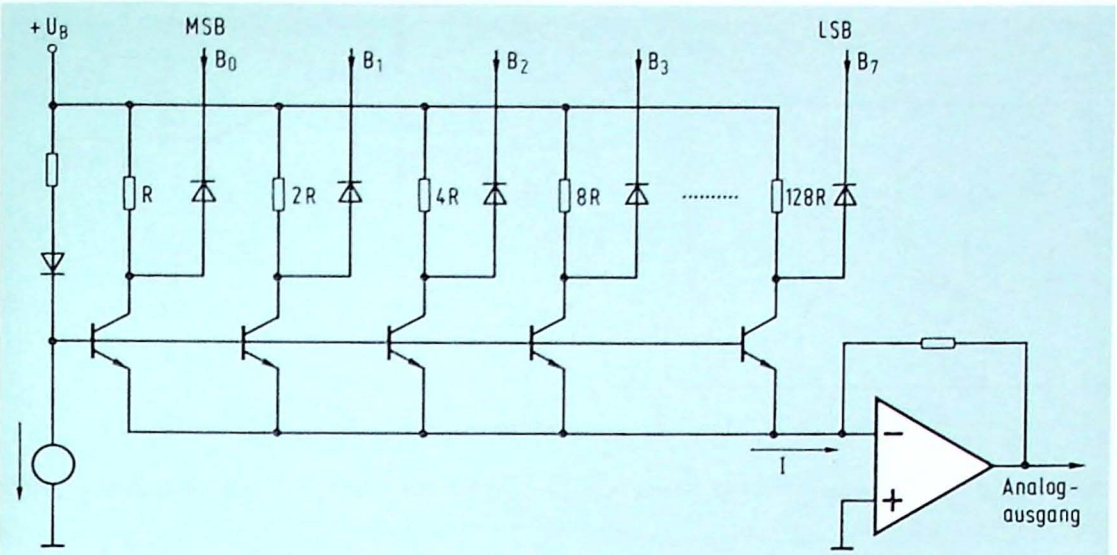


Abb. 1.4.3.1 Realisation eines D/A-Umsetzers mit Widerständen, deren Werte nach Zweierpotenzen aufsteigend gewählt sind

die Helligkeit einer Lampe, die Drehzahl eines Motors oder unterschiedliche Bedienfunktionen eines Fernsehgeräts steuern.

Zu diesem Zweck gibt es sogenannte D/A-Umsetzer. Sie besitzen z.B. acht, zehn oder zwölf digitale Eingänge, die mit einem Ausgangs-Port eines Computers verbunden werden können, und einen analogen Ausgang, an dem je nach angelegtem Datenwort eine stetig veränderliche Spannung abgegriffen werden kann. (Manche D/A-Umsetzer können dank eingebauter Zwischenspeicher-Flipflops auch direkt am Prozessorbus betrieben werden.)

Wie ein D/A-Umsetzer-IC intern aussehen kann, zeigt *Abb. 1.4.3.1* am Beispiel einer 8-Bit-Version. Für jede Bitwertigkeit ist ein entsprechender Widerstand vorhanden, der je nach angelegter Bitkombination den Strom aus einer Konstantstromquelle (links) zum Ausgangsverstärker bestimmt. Dieser Operationsverstärker macht aus dem Strom eine analoge Spannung. Je größer der Widerstand, desto geringer ist sein Einfluß auf den Gesamtstrom am Eingang des Verstärkers. Deshalb ist ein Widerstand der Größe R (z.B. 1 kΩ) dem höchstwertigen Bit (MSB) zugeordnet, während einer mit 128 kΩ = 2↑7 kΩ dem niedrigstwertigen Bit entspricht.

Diese Schaltung hat leider einen gravierenden Nachteil: Es ist technisch sehr schwierig, Widerstände mit derart unterschiedlichen Größen in einem IC mit hinreichender Präzision zu integrieren. Meist wird in der Praxis deshalb eine Schaltung wie in *Abb. 1.4.3.2* verwendet. Sie kommt mit nur zwei Größen von Widerständen aus, nämlich R (z.B. 1 kΩ) und 2R (z.B. 2 kΩ). Diese hinreichend genau zu integrieren, ist weit weniger problematisch. Zur digitalen Ansteuerung werden elektronische Schalter verwendet, die mit Transistoren (bipolar oder FET) aufge-

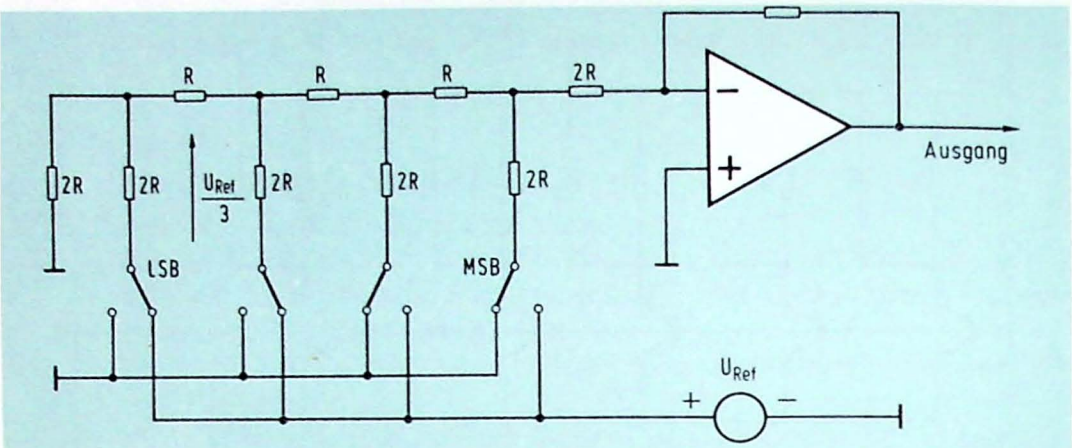


Abb. 1.4.3.2 D/A-Umsetzer mit R-2R-Netzwerk. Dies ist leichter integrierbar als die Schaltung nach Abb. 1.4.3.1

baut sind. Das Beispiel zeigt einen D/A-Wandler mit 4 Bit Auflösung; das Prinzip läßt sich aber leicht auf acht oder mehr Bits erweitern.

Die Qualität von D/A-Umsetzern wird im wesentlichen von drei Faktoren bestimmt:

1. Zahl der Bits am Eingang. Die unvermeidbare Abweichung von einem Wert, der genau zwischen zwei digital reproduzierbaren Ausgangswerten liegt, beträgt 0,5 Uref × 1/2↑n.
2. Genauigkeit der Ausgangsspannung. Sie hängt von der Referenzquelle sowie von der Präzision des Widerstands-Netzwerks ab. Gewöhnlich versucht man, auf „0,5 LSB" zu kommen; das heißt, die Ungenauigkeit soll maximal die Hälfte der Spannungsänderung sein, die beim Umschalten des niederwertigsten Bits auftritt. Zusammen mit dem unvermeidbaren Fehler nach Punkt 1 ergibt sich dann ein maximaler Fehler von Uref/2↑n.
3. Die Einschwingzeit. Bedingt durch interne Kapazitäten und Signallaufzeiten erreicht der Ausgang nicht augenblicklich nach dem Anlegen der entsprechenden Bitkombination seinen Sollwert, sondern erst einige Nano- oder Mikrosekunden später (Settling Time, *Abb. 1.4.3.3*). Die erste Phase der Annäherung an den neuen Wert bezeichnet man als „Slewing", die dabei erreichbare Spannungsänderung pro Zeiteinheit (V/s) als „Slew Rate". Je nach Beschaltung des D/A-Umsetzers kann die Ausgangsspannung kurzfristig auch den Sollwert überschreiten (Überschwingen). In der Mikroprozessor-Technik genügt meist eine Einschwingzeit von einigen Mikrosekunden, da die CPU selbst ja etwa auch diese Größenordnung von Zeit benötigt, um einen neuen Digitalwert auszugeben.

Abb. 1.4.3.4 zeigt einige handelsübliche D/A-Wandler, die nach dem oben beschriebenen Prinzip arbeiten. Mit einem wesentlich geringeren Schaltungsaufwand kommt man dagegen aus, wenn nur geringe Anforderungen an die Geschwindig-

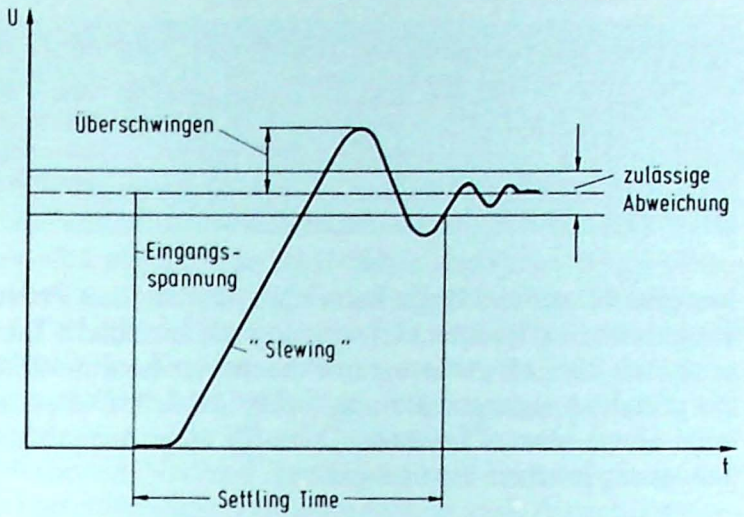


Abb. 1.4.3.3
Charakterisierung des Zeitverhaltens von D/A-Umsetzern

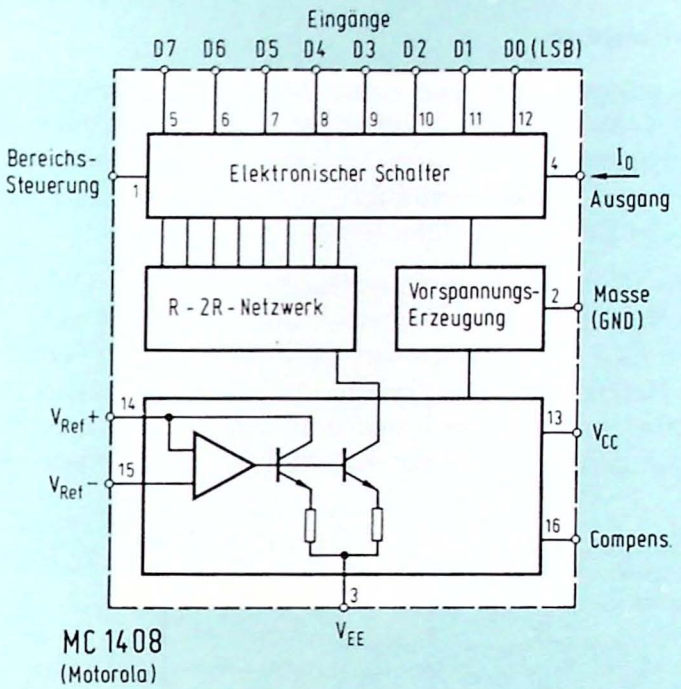
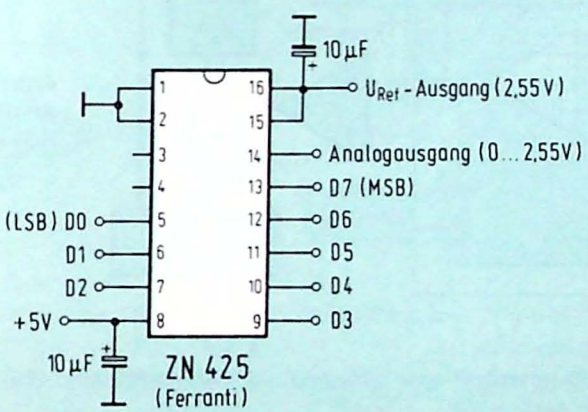


Abb. 1.4.3.4
Anschlußbilder der integrierten D/A-Umsetzer MC 1408 von Motorola und ZN 425 von Ferranti



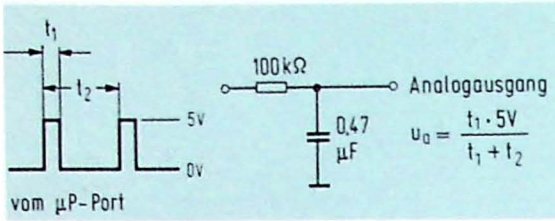


Abb. 1.4.3.5
Prinzip der D/A-Umsetzung durch Integration einer unsymmetrischen Rechteck-Schwingung

keit gestellt werden: Dann kann man nämlich den Prozessor dazu veranlassen, per Programm eine Rechteckschwingung mit variablem Tastverhältnis auszugeben. Ein einfacher Tiefpaß als Integrator macht daraus eine dem Tastverhältnis direkt proportionale Ausgangsspannung *(Abb. 1.4.3.5)*. Die so erreichbare Umsetzzeit beträgt einige zehntel Sekunden, was für viele Aufgaben ausreicht (z.B. Lautstärke-Steuerung in einem Fernsehgerät).

A/D-Umsetzer

Um analoge Signale mit einem Mikrocomputer verarbeiten zu können, benötigt man Analog-Digital-Umsetzer. Mit ihnen kann man ziemlich alles erfassen, was sich irgendwie in eine stetig veränderliche Spannung umsetzen läßt, z.B. Temperatur, Druck, Gaskonzentration, magnetische Feldstärke und vieles andere (hierzu mehr im Kapitel über Sensoren).

Die Vielfalt an Funktionsprinzipien ist bei A/D-Umsetzern noch größer als bei D/A-Wandlern. Von Digitalvoltmetern her bekannt ist das Sägezahn-Verfahren *(Abb. 1.4.3.6)*: Gemessen wird die Zeit, die verstreicht, bis der Kondensator C über eine Referenzspannungsquelle und den Widerstand R auf die gleiche Spannung aufgeladen ist, die am Eingang anliegt. Die Zeitmessung erfolgt digital mit einem Zähler, der mit einem konstanten Takt als Zeitbasis angesteuert wird und an des-

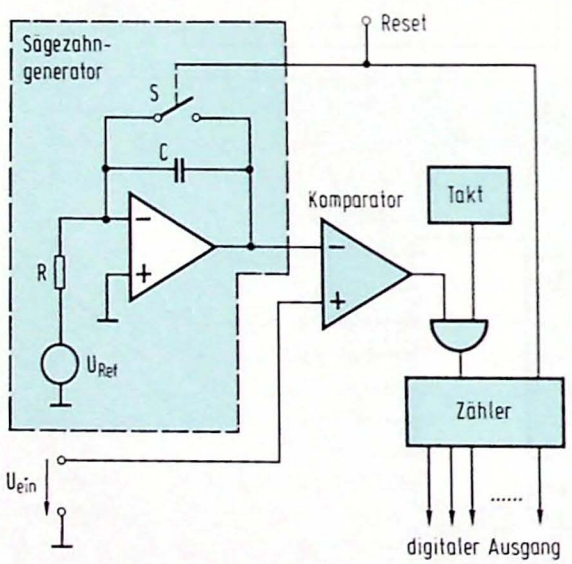


Abb. 1.4.3.6
Analog/Digital-Umsetzung nach dem Sägezahn-Verfahren

sen Ausgang am Ende des Meßzyklus das gewünschte Binärwort anliegt. Zu Beginn einer Messung wird der Kondensator C mit einem Reset-Impuls entladen und der Zähler auf Null zurückgestellt. Leider ist dieses Verfahren sehr anfällig gegenüber Bauelemente-Toleranzen. Insbesondere können schon geringe alterungs- und temperaturbedingte Wertänderungen von R und C das Ergebnis verfälschen.

Besser geeignet ist daher das sogenannte Dual-Slope-Verfahren (Doppelflanken-Verfahren), dessen Meßzyklus in *Abb. 1.4.3.7* wiedergegeben ist. Das Prinzipschaltbild geht aus *Abb. 1.4.3.8* hervor. Der Unterschied zum Sägezahn-Verfahren ist, daß nun auch die Entladezeit des Kondensators gemessen wird. Das Verhältnis von Lade- zu Entladezeit entspricht dem Verhältnis der Eingangsspannung zur Referenzspannung. Die Absolutwerte von R und C gehen nicht mehr in die Messung ein. Der einzige verbleibende Nachteil des Dual-Slope-Verfahrens ist, daß sich keine sehr hohe Umsetzgeschwindigkeit erzielen läßt; je nach Auslegung dauert ein Meßzyklus etwa 0,1 ... 1 Sekunde.

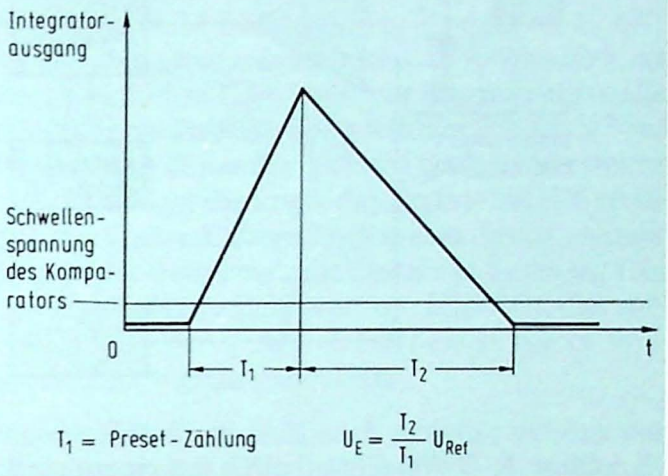


Abb. 1.4.3.7
Das Doppelflanken-Verfahren vermeidet den Einfluß von Toleranzen der zeitbestimmenden Bauelemente im A/D-Umsetzer

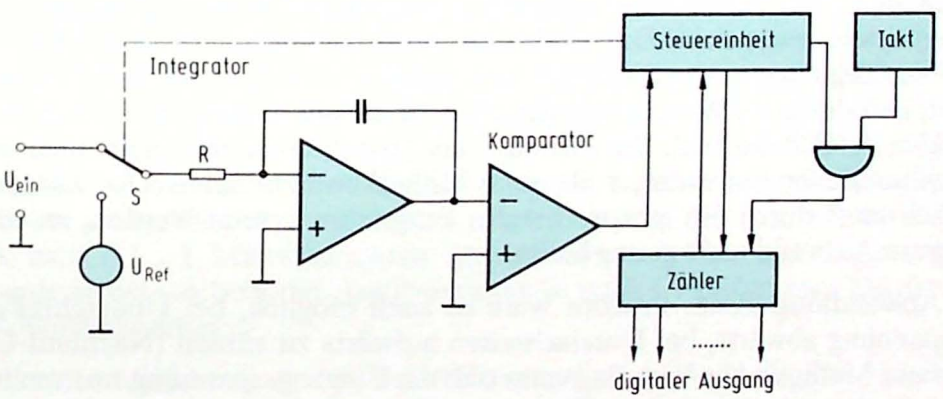


Abb. 1.4.3.8 Blockschaltbild eines A/D-Umsetzers nach dem Doppelflanken-Verfahren (Dual Slope)

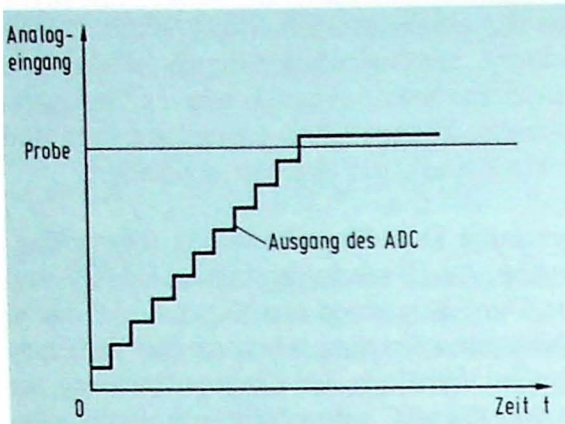


Abb. 1.4.3.9
A/D-Wandler mit eingebautem D/A-Umsetzer erhöhen mit Hilfe eines Zählers den Digitalwert so lange, bis die Ausgangsspannung des D/A-Umsetzers die zu messende Eingangsspannung erreicht hat

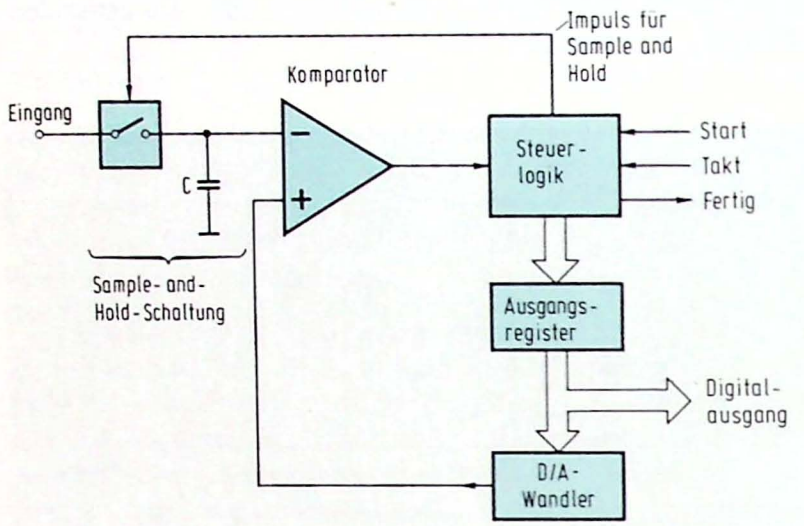


Abb. 1.4.3.10
Blockschaltbild eines A/D-Umsetzers für sukzessive Approximation

Andere A/D-Wandler arbeiten mit einem eingebauten D/A-Umsetzer. Der dabei zugrundeliegende Gedanke ist, das am D/A-Wandler anliegende Digitalwort solange zu ändern, bis seine Ausgangsspannung genau der zu messenden Eingangsspannung entspricht. Das einfachste und naheliegendste Verfahren hierbei ist, dieses Digitalwort von Null solange aufwärts zu zählen, bis ein Komparator meldet, daß der D/A-Wandler-Ausgang die zu messende Spannung überschritten hat *(Abb. 1.4.3.9)*. Je nach der „Breite" des D/A-Umsetzers (in Bit) lassen sich Meßzyklus-Zeiten von weniger als einer Millisekunde erzielen. Der Zähler kann natürlich auch durch ein entsprechendes Programm ersetzt werden, so daß der Hardware-Aufwand recht gering bleibt.

In Abwandlung dieses Prinzips wäre es auch möglich, bei Überschreiten der Meßspannung abwärts, bei Unterschreiten aufwärts zu zählen (Nachlauf-Umsetzer); diese Methode hat Vorteile, wenn sich die Eingangsspannung nur um irgendeinen Mittelwert herum ändert. Die mittlere Umsetzzeit ist dann deutlich geringer als die maximale, die bei einem Spannungssprung voller Höhe auftreten würde.

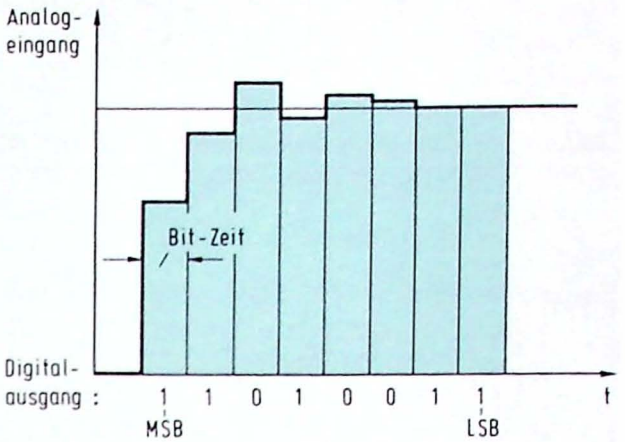


Abb. 1.4.3.11
Mit sukzessiver Approximation sind nur so viele Schritte zur Bestimmung des Eingangswertes nötig, wie der A/D-Umsetzer Bits umfaßt

Recht verbreitet ist das Verfahren der „Sukzessiven Approximation" (schrittweise Annäherung). Der Trick hierbei ist, daß der Zähler durch eine intelligente Logik ersetzt wird, die nacheinander durchprobiert, welche Bits am D/A-Umsetzer 0 und 1 sein müssen, damit möglichst schnell die Meßspannung erreicht wird *(Abb. 1.4.3.10)*. Ein Beispiel: Die maximale Ausgangsspannung des D/A-Wandlers sei 2,55 V, die zu messende Spannung 1,292 V. Am D/A-Umsetzer liegen zunächst alle Bits auf Null. Nun legt die Steuerlogik versuchsweise das höchstwertige Bit D7 auf log. 1, stellt über den Komparator fest, daß damit der Sollwert noch nicht erreicht ist und läßt dieses Bit also auf log. 1. Dann legt die Logik das nächste Bit (D6) probeweise auf 1. Jetzt wird allerdings der Sollwert überschritten, und deshalb nimmt sie D6 wieder auf Null zurück. So geht das weiter bis zum niederwertigsten Bit D0, das noch gesetzt werden muß. Das resultierende Binärwort ist 10000001, das entspricht dezimal genau 129 (entsprechend 1,29 V). *Abb. 1.4.3.11* macht diesen Vorgang bei einer Eingangsspannung von 2,11 V grafisch deutlich.

Eine Besonderheit am Eingang der Schaltung muß noch erwähnt werden: ein elektronischer Schalter und ein Kondensator C. Diese „Sample-and-Hold"-Schaltung wird benötigt, um sicherzustellen, daß sich die Spannung am Eingang des Komparators während des Meßzyklus nicht ändert, da sonst irreführende Meßergebnisse auftreten können. Der elektronische Schalter ist nur zwischen zwei Meßzyklen leitend, wobei die Eingangsspannung den Kondensator auflädt. Bei genügend hohem Eingangswiderstand des Komparators ändert sich während der sukzessiven Approximation die Spannung an C nicht. Eine solche Halteschaltung ist übrigens bei den anderen A/D-Umsetzverfahren nicht erforderlich. Die Umsetzzeit bei sukzessiver Approximation ist recht gering: Bei einer Hardware-Steuerlogik sind 0,1...1 Mikrosekunden typisch. Überläßt man die Steuerung dem Computer selbst, so benötigt das Programm je nach CPU-Typ und Taktfrequenz 0,1...1 Millisekunden.

Ein letztes Verfahren, das in der Mikrocomputertechnik wegen des geringen Hardware-Aufwandes recht beliebt ist, zeigt *Abb.. 1.4.3.12.* Ein spannungsgesteuerter Oszillator liefert eine Frequenz an den Computer, die der Eingangsspannung

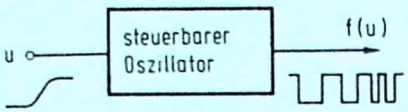


Abb. 1.4.3.12 Eine Messung analoger Werte ist auch mit Hilfe eines steuerbaren Oszillators möglich, dessen Ausgangsfrequenz der Computer per Software messen kann

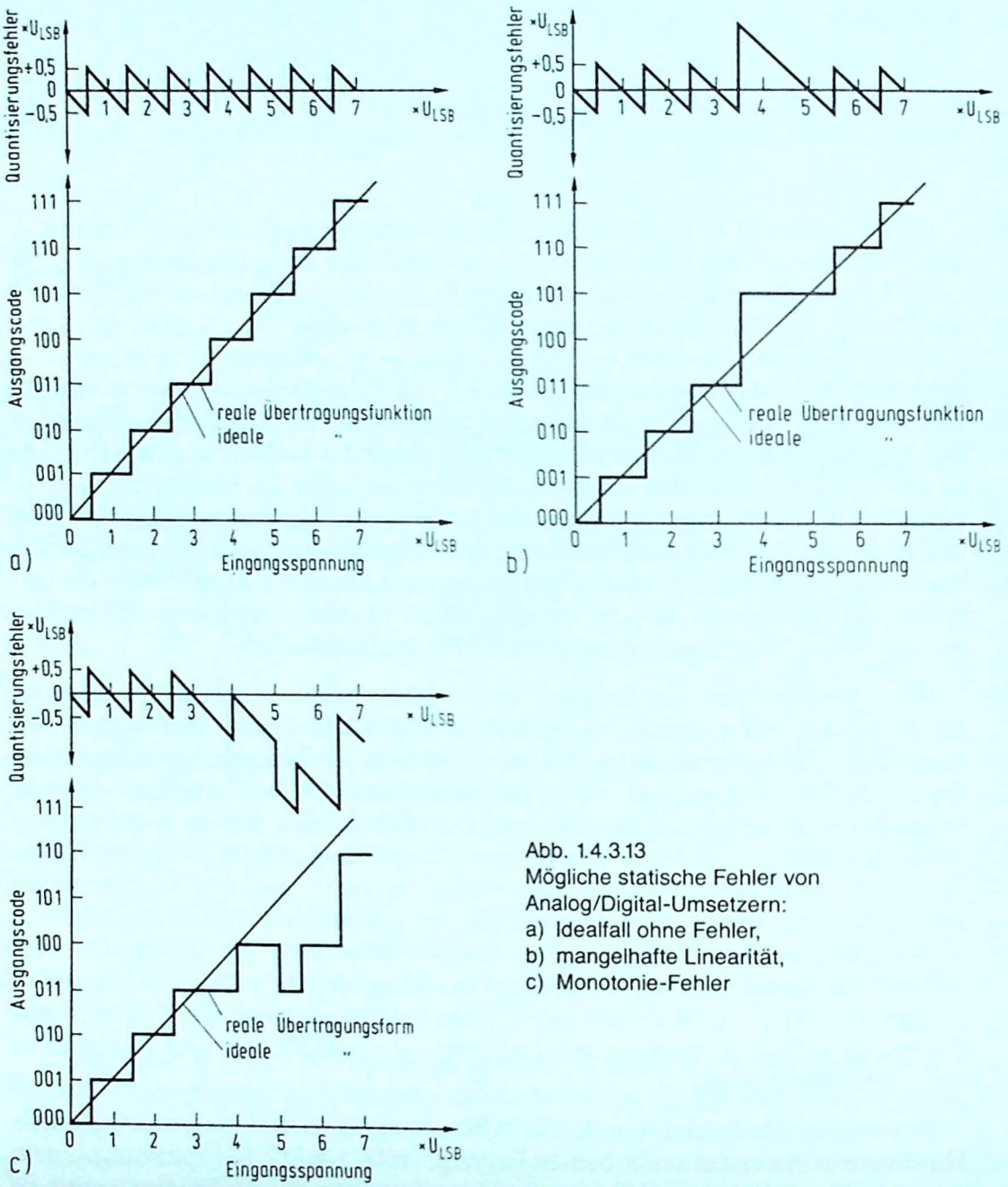


Abb. 1.4.3.13
Mögliche statische Fehler von Analog/Digital-Umsetzern:
a) Idealfall ohne Fehler,
b) mangelhafte Linearität,
c) Monotonie-Fehler

proportional ist und die sich per Programm leicht messen läßt (z.B. 0...10 kHz). Allerdings gehen ähnlich wie beim oben erwähnten Sägezahn-Verfahren Toleranzen der Bauelemente im Oszillator voll in die Meßgenauigkeit ein, und die Messung einer Frequenz durch den Mikrocomputer dauert normalerweise etwa 0,1...1 Sekunde. Alternativ ist es auch möglich, die Periodendauer des Oszillator-Ausgangssignals zu messen, was praktisch auf das Sägezahn-Verfahren hinausläuft, bei dem ja auch die Zeit gemessen wird.

Bei der Beurteilung der Qualität von A/D-Umsetzern gibt es neben der schon erwähnten Umsetzzeit und der durch Bauelemente-Toleranzen bestimmten Genauigkeit noch zwei Kriterien, wie *Abb. 1.4.3.13* zeigt. Wiedergegeben ist zunächst die Übertragungsfunktion eines idealen A/D-Umsetzers. Man erkennt, daß natürlich nicht jeder Zwischenwert darstellbar ist, sondern nur eine bestimmte Anzahl von Werten — je nachdem, aus wie vielen Bits der Ausgangscode besteht.

Probleme bei der Linearität und Monotonie ergeben sich insbesondere bei A/D-Umsetzern, die mit eingebauten D/A-Wandlern arbeiten, welche z.B. im R-2R-Netzwerk Toleranzen aufweisen. Die Hersteller versuchen, Nichtlinearität und Nichtmonotonie geringer als die Wertigkeit des niedrigstwertigen Bits (LSB) zu halten.

1.4.4 Video-Controller

Viele Computer benutzen als Anzeige Video-Monitore oder, bei geringeren Anforderungen an die Schärfe der Darstellung, handelsübliche Fernsehgeräte. Leider sind die hierfür benutzten Normen nicht weltweit einheitlich, wie die folgende Übersicht deutlich macht:

	USA, Kanada	BRD	Großbritannien	Frankreich
Vollbilder	30/s	25/s	25/s	25/s
Vert.-Freq.	60 Hz	50 Hz	50 Hz	50 Hz
Hor.-Freq.	15750 Hz	15625 Hz	15625 Hz	15625 Hz
Zeilen/Bild	525	625	625	625
Farbnorm	NTSC	PAL	PAL	SECAM
Farbträger	3,576 MHz	4,43 MHz	4,43 MHz	4,43 MHz
Ton-Abstand	4,5 MHz	5,5 MHz	6 MHz	6,5 MHz
Bild-Mod.	neg. AM	neg. AM	neg. AM	pos. AM
Ton-Mod.	FM	FM	FM	AM

Weltweit gibt es noch weitere Fernsehnormen, die hier allerdings außer Betracht gelassen werden können.

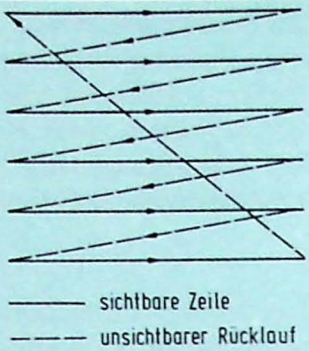


Abb. 1.4.4.1
Die Leuchtschicht des Bildschirms wird zeilenweise von einem Elektronenstrahl beschrieben

Um die oben genannten Daten besser zu verstehen, sind einige Erläuterungen nötig. Ein Fernsehbild wird aus einzelnen horizontalen Zeilen aufgebaut, die von links nach rechts auf den Bildschirm geschrieben werden. Der nur sehr kurzzeitige Rücklauf von rechts nach links bleibt unsichtbar *(Abb. 1.4.4.1)*. Die Zeilen werden nacheinander von oben nach unten geschrieben, und der schreibende Elektronenstrahl kehrt schließlich während des „Bildrücklaufs" von unten rechts nach oben links zurück, um ein neues Bild zu schreiben.

Ein Vollbild besteht (nach deutscher Norm) aus 625 Zeilen und wird 25mal pro Sekunde geschrieben. Leider liegt die Flimmergrenze, also die Trägheit des menschlichen Auges, etwas höher, nämlich bei 30 ...50 Hz, so daß ein unangenehmer Flimmereindruck entstehen würde. Man wendet daher den Trick des „Halbbild-Verfahrens" an: In der ersten $^1/_{50}$ Sekunde schreibt der Elektronenstrahl alle ungeraden Zeilennummern (1, 3, 5, ... 625), in der nächsten fünfzigstel Sekunde alle geraden (2, 4, 6, ... 624). Dadurch ist die Video-Bandbreite einerseits nicht höher als bei 25 Hz Bildfrequenz, andererseits flimmert das Bild normalerweise nicht unangenehm. Viele Video-Controller verzichten allerdings auf dieses Zeilensprung-Verfahren und stellen einfach 50mal pro Sekunde jeweils etwa 312 Zeilen dar, was meist ausreicht.

Preiswerte Heimcomputer verwenden oft den Antennen-Eingang eines Fernsehgeräts, um das auf einen hochfrequenten Träger aufmodulierte Videosignal (AM = Amplitudenmodulation) einzuspeisen. Toneffekte können gleichzeitig über den Lautsprecher des Fernsehgeräts hörbar gemacht werden, wenn der Computer dem Videosignal noch einen frequenzmodulierten Tonträger (FM) hinzufügt, der hierzulande um 5,5 MHz höher als der Bildträger liegt. Die Kanalangaben am Fernsehgerät entsprechen folgenden Frequenzen:

Bereich	Kanal	von—bis/MHz (Bildträger)	Abstand/MHz
VHF/I	1—4	41,25—62,24	7
VHF/III	5—12	175,25—224,25	7
UHF/IV—V	21—69	471,25—855,25	8

(Band II ist hier nicht enthalten, weil dies das UKW-Rundfunkband ist.)

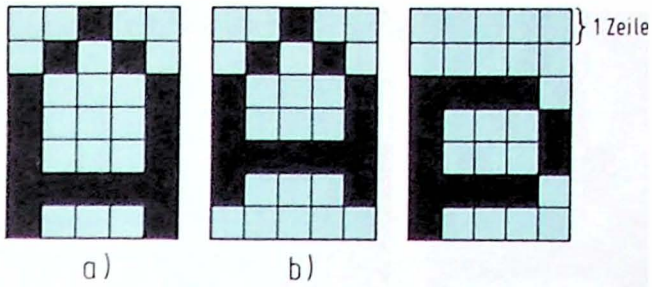


Abb. 1.4.4.2
Schriftdarstellung im 5×7-Punkte-Raster

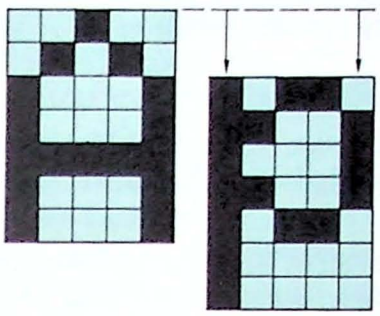


Abb. 1.4.4.3 Wenn das Punktraster vom Video-Controller nach unten verschoben werden kann, ist auch mit 5 × 7 Punkten die Darstellung von Zeichen mit Unterlängen möglich

Die unterschiedlichen Farbfernsehnormen der einzelnen Länder spielen nur bei Einkopplung über den Antenneneingang eine Rolle. Sonst verwendet man bevorzugt den sogenannten RGB-Eingang, der die drei Farbsignale Rot, Grün und Blau getrennt zur Verfügung stellt. Bei Schwarzweiß-Wiedergabe braucht man sich schließlich nur darum zu kümmern, daß der Computer bzw. sein Video-Controller-Baustein die gleichen Ablenkfrequenzen liefert, wie sie der verwendete Monitor verarbeiten kann (50/60 Hz, 15625/15750 Hz). In vielen Fällen liegen diese Unterschiede aber noch innerhalb des Einstellbereichs der Bildfang- bzw. Zeilenfang-Einsteller am Fernsehgerät bzw. Video-Monitor.

Wie werden nun Buchstaben, Ziffern und Satzzeichen auf dem Bildschirm dargestellt? Dazu ist es nötig, diese Zeichen in einzelne Punkte zu zerlegen (Punktmatrix) und diese Matrix dann zeilenweise als Videosignal auszugeben.

Abb. 1.4.4.2 zeigt, wie ein großes A und ein kleines p als 5 × 7-Matrix aussehen können. Vergleicht man das A in Fall a) mit dem p, so sieht man, daß dem p hier die sonst typischen Unterlängen fehlen, wie sie auch bei g, j, q und y vorkommen. Das Problem läßt sich, wie b) zeigt, notdürftig beheben, indem man beim A (und den übrigen Zeichen ohne Unterlängen) auf die unterste Matrixzeile verzichtet. Moderne Video-Controller verfügen aber über eine bessere Möglichkeit, die sich „echte Unterlängen" nennt: Bei der Wiedergabe von g, j, p, q und y wird die Zeichenmatrix um zwei Zeilen nach unten versetzt. Das geht, weil keines der Zeichen mit Unterlängen die obersten zwei Matrixzeilen verwendet *(Abb. 1.4.4.3)*.

Eine verbesserte Darstellungs-Qualität läßt sich erzielen, wenn man die Matrix aus mehr Punkten aufbaut. *Abb. 1.4.4.4* zeigt ein Beispiel mit 7 × 9 Punkten und

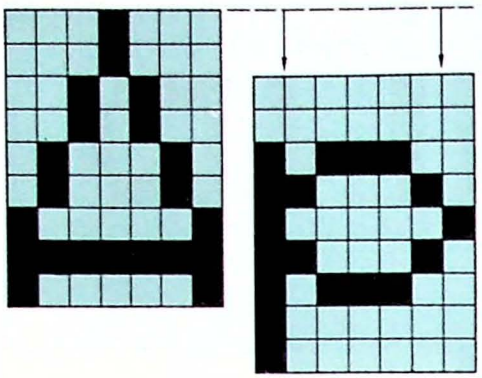


Abb. 1.4.4.4
Mit einer 7×9-Matrix erreicht man schon eine deutlich bessere Darstellungsqualität

wiederum mit dem 2-Zeilen-Versatz bei Unterlängen. Daß sich dieses Spiel nicht beliebig weit treiben läßt, wird aus folgender Überlegung deutlich:

Von den 312 Zeilen eines Bildes (bei 50-Hz-Vollbildern, s.o.) lassen sich maximal etwa 290 tatsächlich ausnutzen, um nicht über den Rand des Bildschirms hinaus zu schreiben. Um zu vermeiden, daß Zeichen mit Unterlängen direkt darunter stehende Großbuchstaben berühren, muß man für ein Zeichen bei einer 7 × 9-Matrix mindestens 9+2+1 Zeile reservieren (1 Zeile als Zwischenraum, 2 als Unterlänge). Demzufolge lassen sich so auf dem Bildschirm maximal 24 Zeichen-Zeilen darstellen (290:12=24,1667).

Tatsächlich verwenden die meisten Computer 24 Zeilen, wobei pro Zeile entweder 22, 32, 40, 64 oder 80 Zeichen dargestellt werden. Für professionelle Geräte kommt allerdings nur ein Controller für 80 Zeichen pro Zeile in Frage, da nur dann auf dem Bildschirm eine typische Schreibmaschinen-Zeile (ca. 70 Zeichen) auch in eine Bildschirm-Zeile paßt.

Geht man von 80 Zeichen pro Zeile bei einer 7 × 9-Punktmatrix für jedes Zeichen aus, so kann man die nötige Video-Bandbreite des Monitors wie folgt berechnen: Zwischen zwei Zeichen sollte horizontal noch ein Punkt als Abstand verbleiben, damit sie sich nicht direkt berühren. Für ein Zeichen muß man also acht Punkte in horizontaler Richtung reservieren. Bei 80 Zeichen pro Zeile sind das 80×8=640 Punkte; allerdings ist am linken und rechten Bildschirmrand noch ein Sicherheitsabstand erforderlich, für den man zusätzlich etwa 2×80=160 Punkte einkalkulieren sollte. Insgesamt sind also horizontal 800 Punkte möglich. Innerhalb der Zeichenmatrix tritt die maximale Video-Frequenz dann auf, wenn helle und dunkle Punkte im Wechsel aufeinander folgen. Die Zeit, in der eine Zeile geschrieben wird, beträgt $^1/_{15625}$ s = 64 µs; ein Punkt dauert daher $^{64}/_{800}$ µs = 80 ns. Ein Hell-Dunkel-Zyklus dauert demnach 2×80 ns = 160 ns; der Kehrwert davon ergibt die nötige Video-Bandbreite, hier 1/160 ns = 12,5 MHz. Damit ist gerade noch eine scharfe Abbildung möglich.

Bei geringerer Zahl von Zeichen pro Zeile ist auch die nötige Bandbreite kleiner:

Zeichen pro Zeile	22	32	40	64	80
Min.Video-Bandbr. (MHz)	3,44	5,0	6,25	10,0	12,5

Da die Bandbreite des Empfangsteils von Fernsehgeräten nur etwa 5 MHz beträgt, ist eine saubere Wiedergabe damit nur bis etwa 32, mit gewissen Qualitätseinbußen noch bis 40 Zeichen pro Zeile möglich. Auch bei direkter Einspeisung in den Video- bzw. RGB-Eingang des Fernsehgeräts kann eine befriedigende Wiedergabe von 80 Zeichen pro Zeile nicht erwartet werden, da der Video-Endverstärker ebenfalls nur etwa 5 MHz schafft. Hier lohnt sich also unbedingt die Anschaffung eines echten Video-Monitors mit einer Bandbreite von wenigstens 15 MHz.

Bei Farbdarstellung kommt noch das Problem hinzu, daß die „Pixel" einer Farbbildröhre nicht beliebig klein gemacht werden können. Herkömmliche Farbröhren sind für das „normale" Fernsehen ausgelegt und zeigen daher schon beim Versuch, 32 Zeichen pro Zeile darzustellen, unschöne Farbränder an den einzelnen Buchstaben. Für professionelle Zwecke gibt es daher spezielle Farb-Datensichtgeräte, bei denen die Farbtripel (rote, grüne und blaue Leuchtstoff-Punkte) näher beieinander liegen. Der Preis allein für eine solche Röhre ist höher als der eines normalen Farbfernsehgeräts. Bei nicht mehr als 40 Zeichen pro Zeile erreicht man allerdings auch mit der neueren Generation von Videotext- und Bildschirmtext-geeigneten Farbfernsehempfängern zufriedenstellende Ergebnisse, wenn man deren RGB- bzw. SCART-Eingang benutzt.

Wie ein Video-Controller-Baustein arbeitet, zeigt *Abb. 1.4.4.5.* In vielen Fällen verwendet man einen Teil des normalen Arbeitsspeichers auch als Bildwiederhol-Speicher: Aus ihm liest der Controller zyklisch alle fünfzigstel Sekunden die ASCII-Zeichen aus und übergibt sie dem Zeichengenerator. Das ist, vereinfacht

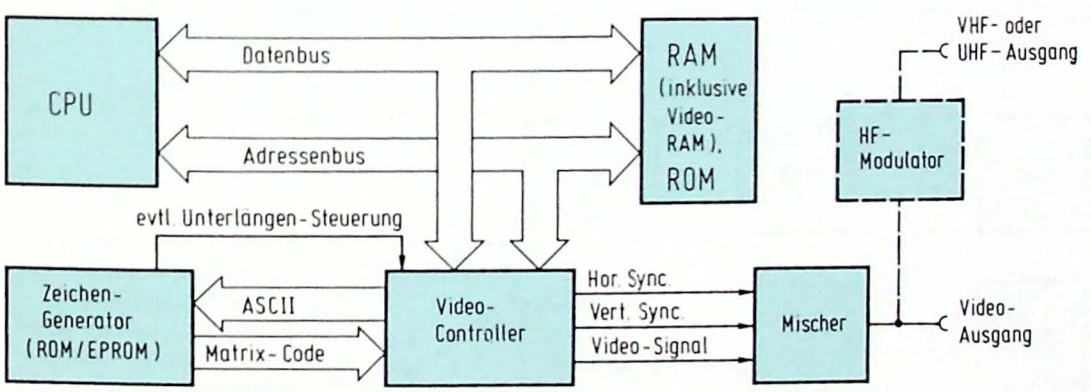


Abb. 1.4.4.5 Prinzip der Zeichenerzeugung in einem Tischcomputer. Der Zeichengenerator, meist ein ROM oder EPROM, enthält die Matrix für jedes darstellbare Zeichen als Bitfolge

gesagt, ein ROM oder EPROM, das an seinen Datenleitungen den Bildschirm-Punktmatrix-Code eines Zeichens liefert, wenn man es als ASCII-Zeichen an seine Adressenleitungen anlegt. Der Video-Controller gibt nun den Matrixcode zeilenweise über ein schnelles Schieberegister aus und kümmert sich gleichzeitig darum, daß am Zeilenende ein Zeilen- und am Bildende ein Bild-Synchronisations-Impuls eingefügt wird. Video- und Synchronsignale werden schließlich in einem Mischer zusammengeführt und entweder direkt dem Monitor oder, über einen HF-Modulator, dem Fernsehgerät zugeführt.

Die Methode, einen Teil des Arbeitsspeichers eines Computers gleich als Bildwiederholspeicher auszulegen, hat einen großen Vorteil: Man kann dafür nämlich dynamische RAM-Bausteine verwenden, ohne eine besondere Refresh-Schaltung zu benötigen. Denn der Video-Controller sorgt bei geeigneter Schaltungsauslegung dank seines zyklischen RAM-Zugriffs selbst für das Auffrischen der Speicherzellen.

Der Vorteil eines gemeinsamen Arbeits- und Bildspeichers ist natürlich, daß die CPU den Bildschirm wie ein normales RAM schreibend und lesend adressieren kann. Dem steht als Nachteil gegenüber, daß die üblichen 8-Bit-Prozessoren nur 64 KByte adressieren können, wovon ein Teil für den Bildwiederholspeicher verloren ginge.

Aus diesem Dilemma gibt es zwei Auswege. Der erste ist, dem Video-Controller ein eigenes RAM zuzuordnen, also nicht über den Adressen- und Datenbus mit der CPU zu verbinden. Der Nachteil hierbei ist natürlich, daß der Datentransfer von der CPU zum Videoteil nicht mit der hohen Busgeschwindigkeit, sondern nur verhältnismäßig langsam über eine Schnittstelle abgewickelt werden kann. Ferner ist zur Realisation dieser Schnittstelle sowohl auf seiten des Video-Controllers als auch des Rechners zusätzliche Hardware nötig *(Abb. 1.4.4.6)*. Diese Methode wird vorzugsweise bei Rechnern mit dem Betriebssystem CP/M angewendet, da dies speziell für diese Art von Datentransfer ausgelegt wurde, also „terminal-orientiert" ist. Auch ist es (wie bei der folgenden Methode) für die CPU nicht ohne weiteres möglich, Zeichen wieder aus dem Bildspeicher zurückzulesen.

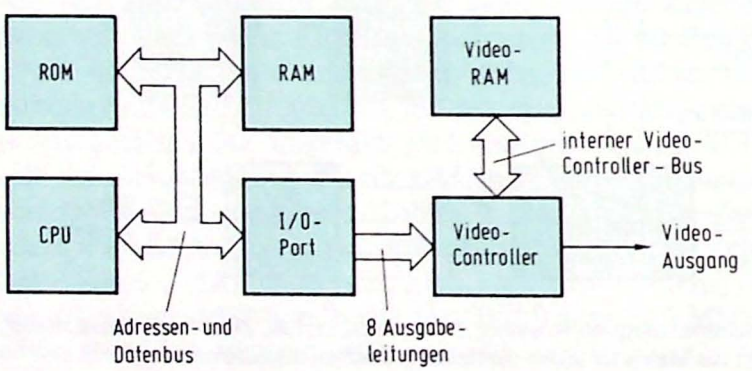


Abb. 1.4.4.6
In manchen Computern ist dem Video-Controller ein eigenes Video-RAM zugeordnet, so daß es im Arbeitsspeicher keinen Adressenraum wegnimmt

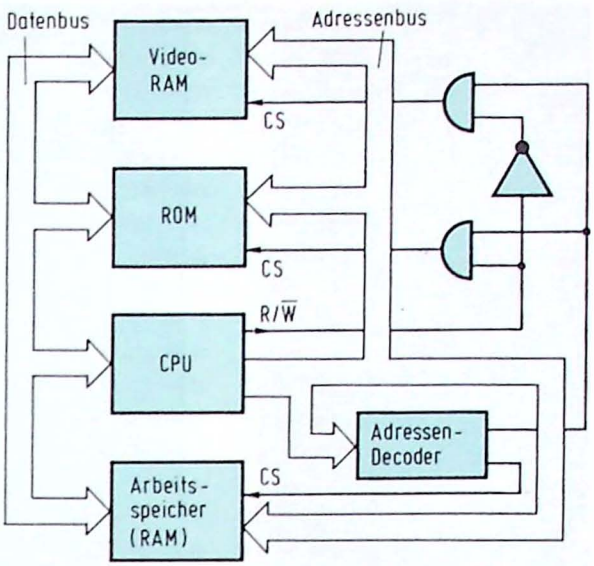


Abb. 1.4.4.7
Mit einem Trick ist es möglich, ROM und Video-RAM in den gleichen Adressenbereich zu legen; zur Umschaltung dient die R/W-Leitung des Prozessors

Die zweite Möglichkeit ist nur bei Computern sinnvoll, bei denen ein Teil des CPU-Adressenraums durch ROMs mit dem Betriebsprogramm (z.B. Basic-Interpreter u.a.) belegt ist. Dabei wird schaltungstechnisch dafür gesorgt, daß bei einem Lesevorgang stets das ROM, bei einem Schreibvorgang dagegen das Video-RAM (Bildwiederholspeicher) angesprochen wird. Die Umschaltung erfolgt also über die R/W-Leitung der CPU *(Abb. 1.4.4.7)*. Nachteilig ist hier, daß aus dem Video-RAM grundsätzlich nicht gelesen werden kann, was programmtechnisch manchmal Vorteile bieten würde, z.B. bei Bildschirm-Editoren.

Ein weiterer Nachteil eines Video-RAM innerhalb des CPU-Adressenbereiches kann sich je nach CPU-Timing ergeben: Es muß ja dafür gesorgt werden, daß sich der Mikroprozessor und der Videocontroller beim Zugriff auf den Arbeitsspeicher nicht gegenseitig stören. Bei einer hohen CPU-Taktfrequenz kann es erforderlich sein, den Prozessor jedesmal kurz anzuhalten, wenn der Video-Controller auf das RAM zugreift. Besonders eklatant ist der Geschwindigkeitsverlust dabei bei ohnehin schon schnellen CPU-Typen; so muß man bei einem 68000-System, das nach diesem Prinzip arbeitet (Anhalten der CPU über den WAIT-Eingang) mit einer Geschwindigkeitseinbuße von etwa 30% rechnen.

Ein besonderes Geschwindigkeitsproblem stellt die Ausgabe von Grafik dar. Beispielsweise müssen zum Zeichnen einer Linie für jeden Punkt zwischen den beiden vorher definierten Endpunkten die Koordinaten errechnet werden. Um die System-CPU von solchen Routine-Aufgaben zu entlasten, wird in hochwertigen Computern oft ein eigener Grafik-Prozessor eingesetzt. So kann z.B. der Typ EF 9366 in einer Sekunde rund eine Million Bildpunkte selbständig zeichnen, wenn ihm nur die Endpunkte von Linien übermittelt werden *(Abb. 1.4.4.8)*.

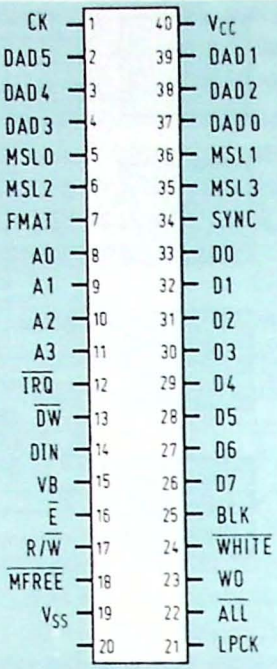


Abb. 1.4.4.8
Anschlußschema der Grafik-Controller-Chips EF 9365/9366

1.4.5 Floppy-Disk-Controller

Die Floppy-Disk (deutsch „Schlappscheibe") ist das heute am weitesten verbreitete Speichermedium für Mikrocomputer; in Kapitel 1.8.1 wird darauf noch näher eingegangen werden.

Als Interface zwischen dem Mikrocomputer-Daten- und Adressenbus und dem Floppy-Laufwerk dienen spezielle Controller-ICs. Oft verwendet werden die Floppy-Controller der Serie WD 179X von Western Digital; sie werden u.a. auch von Siemens gebaut. Die Typen 1795 und 1797 sind auch für zweiseitige Laufwerke geeignet (d.h. Laufwerke, die auf beiden Seiten der Floppy je einen Kopf besitzen). Die Bausteine 1791 und 1795 besitzen einen invertiert herausgeführten Datenbus, bei den Typen 1793 und 1797 ist er nicht invertiert.

Aus *Abb. 1.4.5* geht hervor, daß der Controller einen eigenen Takteingang besitzt (RCLK). Das bedeutet, daß außer dem Controller-IC noch eine Schaltung für die Taktrückgewinnung aus dem empfangenen Signal nötig ist, deren Auslegung in erster Linie von dem verwendeten Aufzeichnungs-Verfahren abhängt (FM, MFM, GCR). Auch darauf kommen wir in Kapitel 1.8.1 noch zu sprechen.

Die seriellen Daten müssen am Controller bereits mit TTL-Pegel anliegen. Deshalb ist auch ein Leseverstärker nötig, der die schwachen Signale vom Floppy-Schreib/Lesekopf von Amplitudenschwankungen befreit und auf 0...5 V begrenzt. Dieser Verstärker ist normalerweise bereits auf einer eigenen Platine (ge-

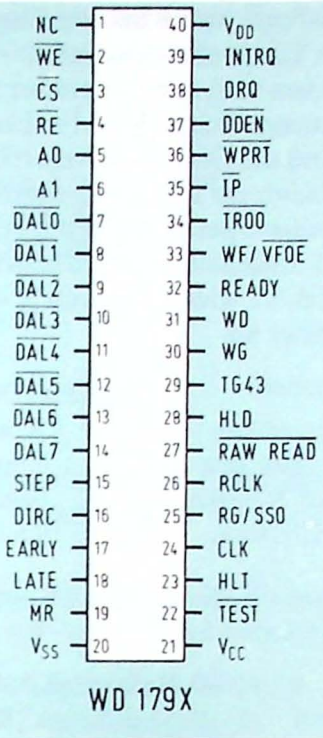


Abb. 1.4.5
Anschlußschema der Floppy-Controller-Chips der Serie 179X von Western Digital

druckte Schaltung) im Floppy-Laufwerk untergebracht, auf der sich auch die Logik zur Ansteuerung der Motoren für Kopfarm und Floppy-Mittelloch befindet.

Doch nun zu den einzelnen Anschlüssen des Controller-IC. DIRC, STEP, WD und WG sind aktiv high und müssen zum Anschluß an das Laufwerk meist invertiert werden. Das kann z.B. mit einem 7406-TTL-Baustein geschehen. DIRC gibt die Richtung an, in die der Kopf bewegt werden soll (High = zur Mitte, Low = nach außen). STEP liefert einen Impuls für jede Spur, um die der Kopf weiterbewegt werden soll. WD liefert die Schreibimpulse (Takt und Daten) für den Kopf; die Schreibfreigabe erfolgt, indem WG auf High-Pegel geht. IP, TR00 und WPRT werden vom Laufwerk bereitgestellt: IP (Indexpuls) gibt den physikalischen Spurbeginn an. TR00 (Track 0) meldet, daß der Kopf über der innersten Spur steht. WPRT zeigt an, daß die eingelegte Diskette mit einem Schreibschutz-Klebestreifen versehen ist, und der Controller führt dann keine Schreiboperation aus.

RCLK ist der zurückgewonnene Takt und RAW READ die mit ihm synchron gelesenen Datenimpulse. VFOE/WF kann als Ein- oder Ausgang arbeiten; bei WG = High meldet WF (Write Fault) einen Schreibfehler. Andernfalls dient VFOE zur Synchronisation der Lesezyklen mit RAW READ.

Über HLT kann das Laufwerk dem Controller mitteilen, daß der Kopf sich stabilisiert hat; allerdings verfügen nicht alle Laufwerke über einen solchen Ausgang. Dann muß man durch eine Verzögerungszeit von etwa 50...100 ms dafür sorgen, daß nicht zu früh gelesen oder geschrieben wird (z.B. mit einem Monoflop). HLD

führt kurz High-Pegel, wenn auf die Floppy zugegriffen wird (z.B. zur Gewinnung eines Signals zur Laufwerks-Auswahl).

Mit DDEN kann man den Controller zwischen einfacher (FM = High) und doppelter Aufzeichnungsdichte (MFM = Low) umschalten.

DAL 0...7 sind die Datenbus-Anschlüsse des Controllers. Über DRQ wird dem Prozessor gemeldet, ob Daten bereitzustellen bzw. abzuholen sind. INTRQ wird aktiviert, wenn eine Operation beendet oder ein Fehler aufgetreten ist. Die Anwahl des Controllers innerhalb des Prozessor-Adressenbereichs erfolgt über den CS-Eingang. A0 und A1 wählen je nach Schreib- oder Lese-Zustand unterschiedliche Controller-Register an:

A1	A0	Schreiben	Lesen
0	0	Statusregister	Kommandoregister
0	1	Spur-Register	Spur-Register
1	0	Sektor-Register	Sektor-Register
1	1	Datenregister	Datenregister

Die Kommandos, die die CPU ins Kommandoregister schreiben kann, haben folgendes Format (Reihenfolge höchst- bis niederwertigstes Bit):

0	0	0	0	h	v	r1	r0	Restore (Kopf auf Spur 0 stellen)
0	0	0	1	h	v	r1	r0	Seek Track (Kopf auf gewünschte Spur)
1	0	0	m	f2	e	f1	0	Read Sector (Sektor lesen)
1	0	1	m	f2	e	f1	a0	Write Sector (Sektor schreiben)
1	1	1	0	0	e	0	0	Read Track (Spur lesen)
1	1	1	1	0	e	0	0	Write Track (Spur schreiben)
0	0	1	u	h	v	r1	r0	Step (Kopf um eine Spur bewegen, Richtung wie im letzten Kommando)
0	1	0	u	h	v	r1	r0	Step in (Kopf nach innen bewegen)
0	1	1	u	h	v	r1	r0	Step out (Kopf nach außen bewegen)
1	1	0	0	0	e	0	0	ID-Feld lesen (s. Kap. 1.8.1)
1	1	0	1	i3	i2	i1	i0	Force Interrupt (Interrupt auslösen)

Die mit Buchstaben bezeichneten Bits bedeuten:

r0, r1	Kopf-Schrittgeschwindigkeit = 6/12/20/30 ms
h	Head-Load-Flag
v	Verify-Flag
f	Flag f. Prüfung der Disk-Seitennummer
e	Flag f. 0 oder 15 ms Verzögerung von HLT nach HLD
m	Flag, um mehrere Sektoren nacheinander zu lesen, bis ein Force-Interrupt-Kommando gegeben wird
a0	Flag für Data Address Mark; 0 = gültig (FB), 1 = gelöscht (F8)
i0	Kommando in Arbeit (Flanke)
i1	Kein Kommando in Arbeit (Flanke)
i2	1 bei jedem Index
i3	1 = Abbruch der Durchführung eines Kommandos

Aus dem Statusregister kann die CPU den momentanen Zustand des Controllers bzw. des Laufwerks abfragen. Bei Positionierungskommandos ergibt sich folgende Bedeutung der acht Bits:

Bit	Bedeutung bei Bit = 1:
7	Ready-Eingang invertiert (Laufwerk nicht bereit)
6	WPRT-Eingang invertiert (Schreibschutz)
5	Kopf auf Disk aufgesetzt (aus HLD/HLT)
4	Falsche Spurnummer gelesen
3	CRC-Fehler im ID-Feld festgestellt
2	Kopf steht über Spur 0 (aus TR00)
1	IP-Eingang invertiert (Index-Loch erkannt)
0	Kommando ist noch in Arbeit

Es ist ratsam, nach Ausgabe eines Kommandos an das Kommandoregister noch etwa 30 (MFM) bzw. 60 Mikrosekunden (FM) zu warten, bevor man das Statusregister abfragt.

Bei Schreib- oder Leseoperationen haben die Bits des Statusregisters z.T. eine andere Bedeutung:

Bit	Lesen (Bit=1):	Schreiben (Bit=1):
6	immer 0	Schreibschutz
5	Datenmarke=F8	Schreibfehler
2	Daten zu spät abgeholt	Daten zu spät bereitgestellt
1	Datenregister enthält Daten	Datenregister ist leer

Die übrigen Bits haben die gleiche Bedeutung wie bei der Kopfpositionierung.

Es ist klar, daß der Floppy-Controller noch eine ganze Menge Software braucht, um seine Aufgabe erfüllen zu können. Davon wird in einem eigenen Kapitel über Floppy-Betriebssysteme die Rede sein.

1.4.6 Sonstige Peripherie-Bausteine

Echtzeit-Uhr

Um in einem Mikrocomputer die tatsächliche Uhrzeit (Echtzeit) zur Verfügung zu stellen, gibt es zwei Möglichkeiten. Zum einen kann z.B. von einem Timer-Baustein (Z80-CTC, 6522, 6532 usw.) regelmäßig, also beispielsweise alle 20 ms, ein Interrupt ausgelöst werden. Die CPU springt dann alle 20 ms in eine Interrupt-Routine und stellt dabei mehrere Speicherzellen weiter, die als Zeitzähler dienen. Dabei kann durch eine geeignete Arithmetik dafür gesorgt werden, daß eine Speicherzelle die Stunden, die nächste die Minuten, eine die Sekunden und eine weitere z.B. fünfzigstel Sekunden enthält.

Dieses Verfahren kommt meist ohne zusätzliche Hardware aus und wird daher gern bei preiswerten Computern eingesetzt. Es besitzt aber einen entscheidenden Nachteil: Schaltet man das System aus, so geht der Speicherinhalt und damit die Uhrzeit verloren. Die Software-Uhr müßte daher jedesmal neu gestellt werden, wenn man den Computer einschaltet. Selbst bei Verwendung eines batteriegepufferten CMOS-RAMs bleibt dieses Problem bestehen, weil ja auf jeden Fall bei abgeschaltetem Rechner der Interrupt nicht funktioniert. Ferner dürfte der Interrupt nicht anderweitig verwendet werden (z.B. bei MS-DOS).

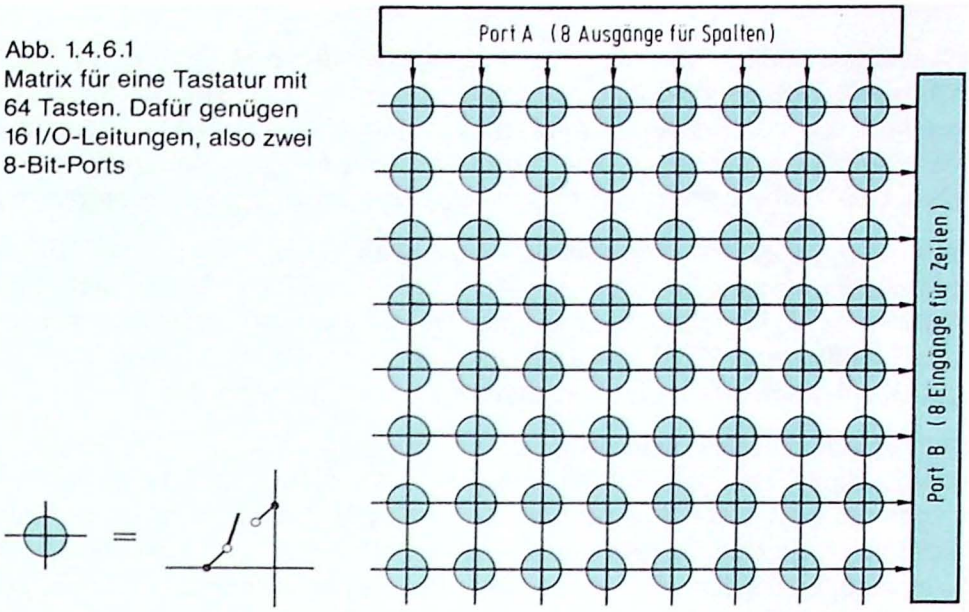
Die Alternative ist eine Hardware-Echtzeit-Uhr. Sie kann aus einer eigenen Batterie versorgt werden, da sie (wie eine LCD-Armbanduhr) sehr wenig Strom benötigt, und braucht daher nicht regelmäßig neu gestellt werden. Der Mikrocomputer kann die Uhr entweder über einen I/O-Port oder, wenn der Uhrenbaustein dafür ausgelegt ist, direkt über den Adressen- oder Datenbus jederzeit die aktuelle Uhrzeit abrufen. Manche Uhrenbausteine liefern außer der Uhrzeit auch noch das Datum und die Nummer des Wochentages. In diesem Fall werden die Zahl der Tage eines Monats und meist sogar auch Schaltjahre automatisch berücksichtigt.

Keyboard-Encoder

Computer besitzen üblicherweise eine Tastatur, die in ihrer Tasten-Anordnung und -Zahl etwa der einer Schreibmaschine entspricht — meist etwa 60. Es ist klar, daß es wenig zweckmäßig ist, für jede Taste eine eigene I/O-Port-Leitung bereitzustellen; das würde ja bedeuten, daß man vier Port-Bausteine mit je 16 I/O-Anschlüssen allein für diesen Zweck „verbraten" müßte. Zweckmäßigerweise verschaltet man deshalb alle Tasten zu einer Matrix *(Abb. 1.4.6.1)*. Legt man jede Matrix-Spalte und -Zeile an einen I/O-Anschluß, so sind bei 64 Tasten nur noch 16 I/O-Leitungen und damit nur ein einziger Port-Baustein nötig. Der Computer legt zunächst an alle Matrixzeilen einen High-Pegel an (log. 1) und schiebt dann ein Null-Bit durch die Zeilen — bis er an einer Spaltenleitung ebenfalls ein Null-Signal feststellt (die Ruhelage ist high, da der Portbaustein intern über Pull-Up-Widerstände verfügt). Der nun in den beiden Port-Datenregistern stehende Code gibt die Position der gedrückten Taste in der Matrix an; er muß nur noch mit Hilfe einer Tabelle in einen 8-Bit-Zeichencode (meist ASCII) umgeformt werden.

Das geschilderte Verfahren ist leider für alle Computer wenig sinnvoll, die mit einer absetzbaren Tastatur ausgestattet sind. Denn das Tastaturkabel müßte dann mindestens 16 Leitungen besitzen, und entsprechend teuer und groß wären die nötigen Steckverbindungen. Deshalb verlagert man die zum Decodieren der Tastenmatrix nötige Intelligenz in das Tastaturgehäuse selbst. Dazu kann ein kleiner 1-Chip-Mikrocomputer oder ein speziell dafür gebauter Logik-Chip, ein Keyboard-Encoder, eingesetzt werden. Für die Verbindung zwischen Tastatur und Computer sind im günstigsten Fall (nämlich bei serieller Übertragung) nur noch

Abb. 1.4.6.1
Matrix für eine Tastatur mit 64 Tasten. Dafür genügen 16 I/O-Leitungen, also zwei 8-Bit-Ports



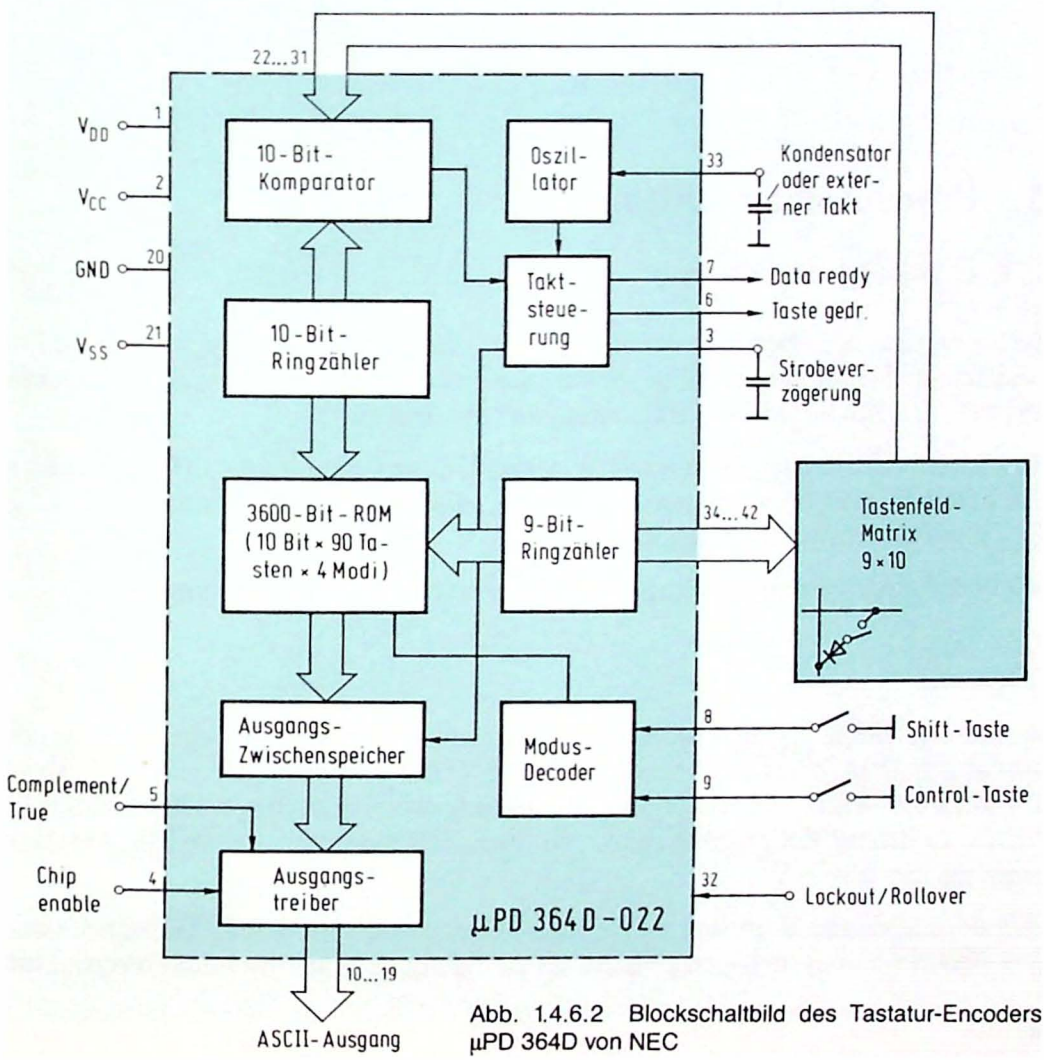


Abb. 1.4.6.2 Blockschaltbild des Tastatur-Encoders µPD 364D von NEC

drei Leitungen nötig — eine Datenleitung, auf der die ASCII-Zeichen Bit für Bit asynchron übertragen werden, eine Masseleitung und eine für die Spannungsversorgung des Keyboard-Encoders. Überträgt man die ASCII-Daten nicht seriell, sondern parallel, so sind insgesamt zehn Leitungen nötig — sieben für die ASCII-Zeichen, eine Strobe-Leitung und zwei weitere für Masse und Spannungsversorgung.

Abb. 1.4.6.2 zeigt den Innenaufbau eines Tastatur-Encoders von NEC, der einen parallelen Ausgang besitzt. Die Shift- und Control-Taste liegen nicht innerhalb der Matrix, da sie ja gleichzeitig mit einer Buchstabentaste gedrückt werden können (z.B. Shift-j = J, CTRL-j = Line-Feed-Steuerzeichen). Die bis zu 90 Tasten sind in einer vom Hersteller festgelegten 9×10-Matrix angeordnet.

Der Anschluß 32 des Bausteins gestattet die Wahl zwischen zwei Betriebsarten: „n-Key-Lockout" bedeutet, daß eine zweite Taste, die nachträglich nach einer ersten gedrückt wird, einfach ignoriert wird. „n-Key-Rollover" heißt, daß diese zweite zusätzlich gedrückte Taste die erste überschreibt. „n-Key" weist darauf hin, daß sich dieses Spiel wiederholt, wenn weitere Tasten mit zeitlicher Überlappung gedrückt werden; beim „2-Key-Rollover" ist das nur für die ersten zwei Mal der Fall.

1.5 Unvermeidbarer „Kleinkram"

1.5.1 Diskrete Bauelemente

Mikrocomputer bestehen im wesentlichen aus elektronischen Bauelementen. Diese werden in „Schaltbildern" oder „Schaltplänen" mit sogenannten „Schaltzeichen" dargestellt, die die Funktion des Bauelements symbolisieren.

Beinhaltet ein Bauelement-Gehäuse tatsächlich nur ein einziges Bauelement im elektrischen Sinne, so spricht man von einem „diskreten" Bauelement. *Abb. 1.5.1.1* zeigt die dazugehörigen Schaltzeichen.

Auf die Funktionen der wichtigsten Bauelemente sei kurz eingegangen.

Die Batterie

Die Batterie dient als Spannungsquelle für netzunabhängige Versorgung. Die Spannung beträgt bei der üblichen (Einweg-)Zink-Kohle-Zelle etwa 1,5 V; bei 1,1 V gilt sie als entladen. Eine höhere Spannung kann man durch Hintereinanderschalten mehrerer Zellen erreichen; mit vier Zellen erreicht man die vierfache Spannung, hier also 6 V.

Wiederaufladbare Batterien nennt man Akkumulatoren. Meist verwendet man dabei Nickel-Cadmium-Typen (NiCd), da sie keinerlei Wartung bedürfen und eine

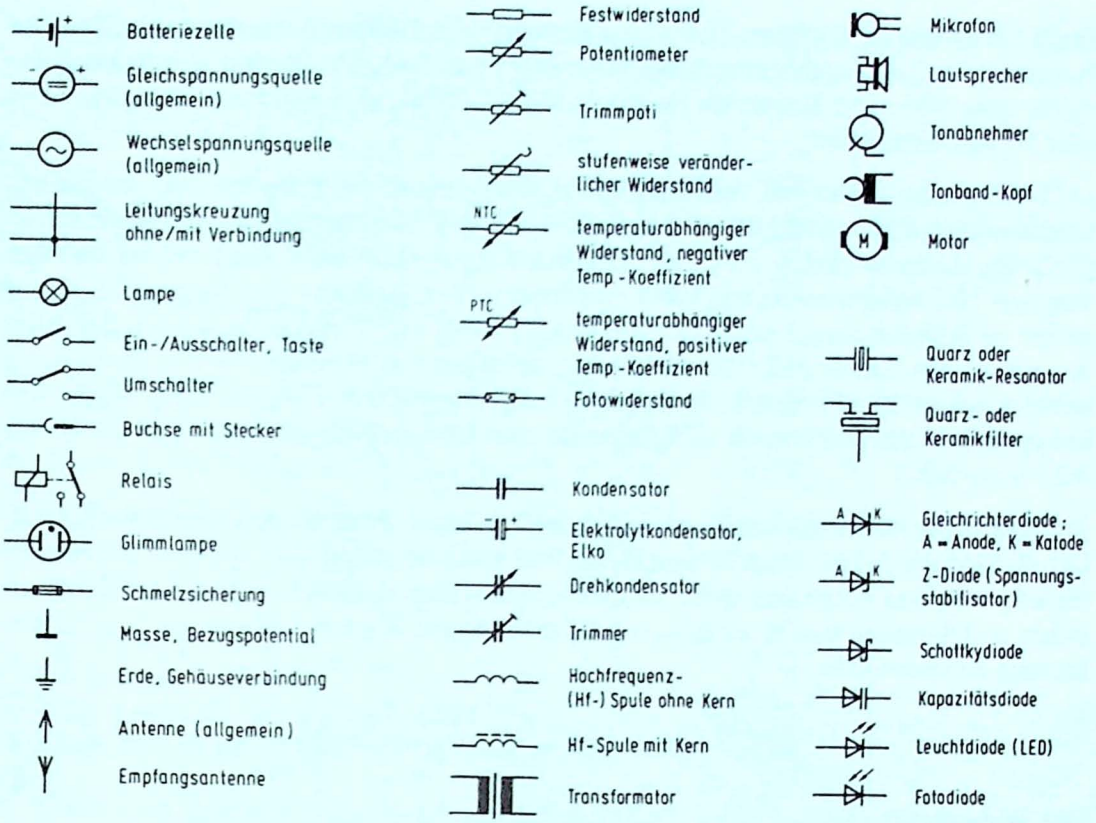


Abb. 1.5.1.1
Die wichtigsten Schaltzeichen der Elektronik

lange Lebensdauer besitzen: Man kann sie einige hundertmal wieder aufladen. Die Kapazität wird in mAh gemessen (Milliampere-Stunden). Besitzt ein Akku beispielsweise 500 mAh Kapazität (Mignon-NiCd-Zelle), so kann man zehn Stunden lang 50 mA entnehmen.

Die Aufladung von NiCd-Akkus sollte man erst dann vornehmen, wenn ihre Spannung vom Nennwert 1,2 V auf etwa 1...1,1 V abgesunken ist (Tiefentladungen sollte man vermeiden, da sie die Lebensdauer verkürzen!). Als Ladestrom wird von den Herstellern meist ein Wert empfohlen, der sich aus der Kapazität geteilt durch 10 Stunden errechnet (z.B. 500 mAh / 10 h = 50 mA). Weil aber der Wirkungsgrad des Akkus nicht 100% beträgt, muß man mit diesem Strom dann nicht zehn, sondern 14 Stunden lang laden. Die Spannung einer Zelle steigt dabei auf knapp 1,5 V an, geht nach Abklemmen des Ladegeräts aber wieder auf etwa 1,25 V zurück.

Sogenannte Sinter-Akkus besitzen speziell gebaute Elektroden, die eine schnellere Aufladung erlauben, z.B. innerhalb von zwei Stunden. Hierbei ist aber empfehlenswert, den Akku nur etwa bis 80% seiner Kapazität mit erhöhtem Strom zu laden und dann auf den Normalwert (siehe oben) zurückzuschalten, um eine Überhitzung zu vermeiden.

Der Widerstand

Widerstände dürften das meistgebrauchte Bauelement der Elektronik sein. Es gibt sie in unterschiedlichen Größen und Belastbarkeiten. Damit die Hersteller nicht eine unvertretbare große Auswahl an Werten bereithalten müssen, hat sich die sogenannte E12-Reihe eingebürgert. Sie heißt so, weil sie innerhalb einer Dekade (d.h. zwischen 1 und 10, zwischen 100 und 1000 usw.) 12 Werte aufweist:

1 1,2 1,5 1,8 2,2 2,7 3,3 3,9 4,7 5,6 6,8 8,2

Die Werte sind so gewählt, daß bei der üblichen Toleranz (zulässige Abweichung vom Sollwert) von +/− 10% gerade noch keine Überschneidung der Toleranzbereiche zweier benachbarter Werte auftritt.

Die Belastbarkeit eines Widerstands wird in Watt angegeben; in einem Mikrocomputer genügt meist ⅛ oder ¼ W. Die tatsächlich vom Widerstand „verbratene" Leistung in Watt errechnet sich aus der anliegenden Spannung und dem durch ihn fließenden Strom:

$P = U*I$ mit P=Leistung/W, U=Spannung/V und I=Strom/A

Der Strom wiederum läßt sich aus der anliegenden Spannung und dem Widerstandswert bestimmen (Ohmsches Gesetz):

$I = U/R$ mit I=Strom/A, U=Spannung/V und R=Widerstand/Ω

Ersetzt man in der ersten Formel den Wert für I durch die rechte Seite der zweiten Gleichung, so ergibt sich als verbrauchte Leistung im Widerstand:

P = U*U/R oder P = U↑2 / R

Da die meisten Widerstände einer elektronischen Schaltung größer sind als 1000 Ω und manche sogar mehr als 100000 Ω besitzen, gibt es noch folgende Einheiten:

1 MΩ = 1000000 Ω (Megohm)
1 kΩ = 1000 Ω (Kiloohm)

In Schaltbildern läßt man aus Bequemlichkeit oft das Ω-Zeichen weg; schließlich ist ja ohnehin klar, daß ein Widerstand in Ω gemessen wird. Die Angabe „100" bedeutet dann 100 Ω, und 4,7 k heißt 4700 Ω.

Üblicherweise ist auf dem Widerstand sein Wert allerdings nicht in Klartext aufgedruckt, sondern in einem speziellen Code aus Farbringen. Bei den üblichen Widerständen der E12-Reihe bedeuten die Ringe:

1. Ring: Erste Ziffer
2. Ring: Zweite Ziffer
3. Ring: Zahl der folgenden Nullen
4. Ring: Toleranz; Silber = 10%, Gold = 5%

Fehlt ein vierter Ring, so beträgt die Toleranz +/− 20%. Der erste Ring ist daran zu erkennen, daß er einem Ende des (meist zylindrischen) Widerstandskörpers am nächsten ist. Die ersten drei Ringe sind folgendermaßen codiert:

schwarz	0	blau	6
braun	1	violett	7
rot	2	grau	8
orange	3	weiß	9
gelb	4		
grün	5		

Beispiel: Bei einem 4700-Ω-Widerstand der E12-Reihe (10% Toleranz) sind die vier Farbringe gelb, violett, rot und silber.

Ist schon der dritte Ring gold- oder silberfarben, so bedeutet das, daß nach den ersten beiden Ziffern keine Null mehr kommt, sondern der Wert mit 0,1 (Gold) oder 0,01 (Silber) zu multiplizieren ist. Bei einem 1,5-Ω-Widerstand sind also die ersten drei Ringe braun, grün und golden (15 * 0,1 = 1,5).

Intern sind die Widerstände gewöhnlich mit einer Kohleschicht oder, wenn es auf hohe Alterungsbeständigkeit ankommt, einem sehr dünnen Metallfilm aufgebaut.

Außer Festwiderständen gibt es noch variable Widerstände. Je nachdem, ob sie nur für eine einmalige Justage (z. B. CPU-Taktfrequenz) oder zur regelmäßigen Be-

dienung gedacht sind (z.B. Kontrasteinstellung an einem Fernsehgerät), bezeichnet man sie als Trimmer oder als Potentiometer. Meist ist eine Kohlebahn in einem nicht ganz geschlossenen Kreis angeordnet, die von einem drehbaren Kohlestift abgegriffen werden kann. Für hohe Genauigkeits-Anforderungen gibt es auch Spindelpotis und -trimmer; bei ihnen ist die Widerstandsbahn gerade, und eine Schraubenspindel bewegt den Abgriff. Die Bahn kann entweder einen linearen oder, z.B. zur gehörmäßig richtigen Lautstärke-Einstellung, einen logarithmischen Verlauf aufweisen.

Der Kondensator

Kondensatoren können elektrische Ladungen speichern. Sie bestehen im Prinzip aus zwei Metallfolien, die voneinander durch eine dünne Isolierschicht, meist aus Kunststoff- oder Keramikmaterial, getrennt sind. Ihre Kapazität wird in Farad (F) gemessen, wobei diese Einheit viel zu groß ist, um tatsächlich verwendet zu werden. Eingeführt sind folgende Bezeichnungen:

1 pF (Picofarad) $= 10^{-12}$ F
1 nF (Nanofarad) $= 10^{-9}$ F
1 µF (Mikrofarad) $= 10^{-6}$ F
1 mF (Millifarad) $= 10^{-3}$ F

In der Computertechnik werden Kondensatoren zum einen als frequenzbestimmende Bauelemente in Taktgeber-Schaltungen (Oszillatoren), zum anderen zum „Abblocken" verwendet. Letzteres bedeutet das Fernhalten von Wechselspannungsanteilen und kurzen Störimpulsen von der Versorgungsspannung: Kurzzeitige Spannungsschwankungen werden sozusagen durch die im Kondensator gespeicherte Ladung aufgefangen. Für Abblockzwecke setzt man Kondensatoren von etwa 0,1 µF aufwärts ein, in Netzgeräten (als Siebkondensatoren gegen den vom Gleichrichter noch übrigen Wechselspannungsanteil) von 1000 µF aufwärts.

Legt man an einen entladenen Kondensator eine Gleichspannung an, so lädt er sich zunächst auf diese Spannung auf, indem er Strom aus der Spannungsquelle entnimmt. Danach stellt er für Gleichstrom einen unendlich großen Widerstand dar, es fließt also kein Strom mehr, solange sich die angelegte Spannung nicht ändert.

Für Wechselstrom ist der scheinbare Innenwiderstand (der „Scheinwiderstand") des Kondensators der Frequenz umgekehrt proportional:

$Z = 1/(2*Pi*C*f)$

Dabei bedeuten:

Z = Scheinwiderstand des Kondensators in Ohm
Pi = Kreiszahl (3,141...)
C = Kapazität des Kondensators in Farad
f = Frequenz der angelegten Wechselspannung

Ein Beispiel: Für die 1-MHz-Taktfrequenz eines Computers (10^6 Hz) stellt ein Kondensator von 0,1 µF einen Widerstand von $Z = 1/(2 * 3{,}14 * 0{,}1 * 10^{-6} * 10^6)\ \Omega = 0{,}159\ \Omega$ dar.

Große Kapazitäten von mehr als 0,1 µF lassen sich mit herkömmlichen Isolationsmaterialien nicht mehr platzsparend realisieren. Dafür verwendet man deshalb sogenannte Elektrolyt-Kondensatoren (Elkos, *Abb. 1.5.1.2*). In dem flüssigen (Natronlauge) oder festen Elektrolyt (Tantal, *Abb. 1.5.1.3*) bildet sich unter Anlegen einer Formierungsspannung während des Herstellprozesses eine sehr dünne, aber isolierende Oxidschicht. Die erreichbare Kapazität ist umso höher, je dünner die Isolierschicht ist, und deren Dicke wird wiederum von der Höhe der Formie-

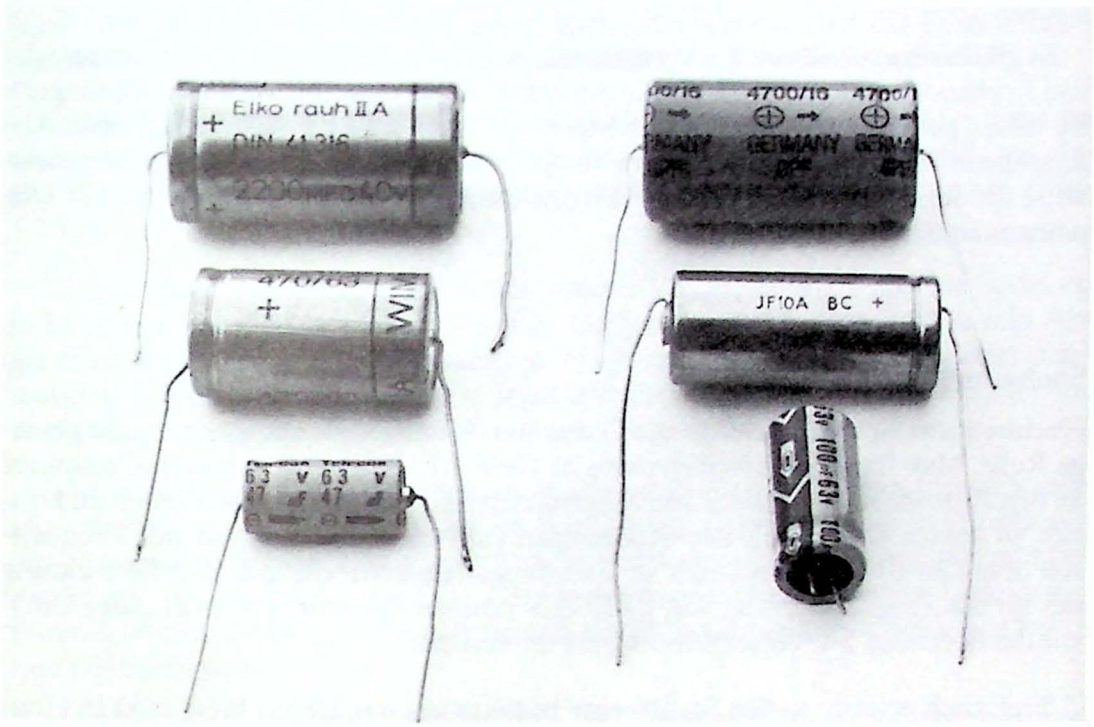


Abb. 1.5.1.2 So sehen Elektrolyt-Kondensatoren mit feuchtem Elektrolyt aus

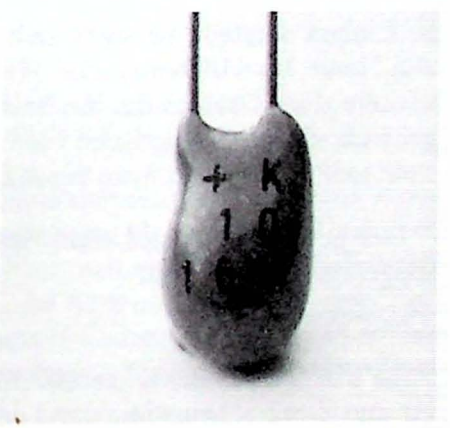


Abb. 1.5.1.3
Tantal-Elkos sind meist in einem tropfenförmigen Gehäuse untergebracht

rungsspannung bestimmt. Allerdings dürfen solche Kondensatoren nur noch mit der gleichen Polarität wie beim Formieren verwendet werden; in der anderen Richtung wirkt die Oxidschicht nämlich nicht isolierend. Elkos sind daher auf einer Seite mit einem Plus-Zeichen markiert.

Bei Kondensatoren hat sich die gleiche E12-Wertereihe eingebürgert wie bei Widerständen. Speziell bei Elkos muß man allerdings mit erheblich größeren Toleranzen als 10% rechnen. Der Sollwert ist meist im Klartext aufgedruckt, allerdings oft unter Weglassung der Einheit. Die Angabe „.01" auf einem keramischen Scheibenkondensator bedeutet 0,01 µF. „100" auf einem kleinen Keramik- oder Kunststoffkondensator bedeutet 100 pF, und „10/6" auf einem Elko kennzeichnet einen Kondensator mit 10 µF und 6 V Spannungsfestigkeit.

Es gibt auch einstellbare Kondensatoren, nämlich Trimmer für Abgleichzwecke und Drehkondensatoren (z.B. für die Abstimmung in einem Rundfunkempfänger). Sie lassen sich aber nur bis zu Kapazitäten von etwa 500 pF bei vertretbaren Abmessungen herstellen. In Computern findet man Trimmkondensatoren beispielsweise für die exakte Justierung des Taktoszillators für die CPU selbst und für Uhrenbausteine.

Spulen und Transformatoren

Hochfrequenz-Spulen spielen in der Computer-Schaltungstechnik eine recht geringe Rolle. Man findet sie beispielsweise in UHF-Modulatoren, um den Computer an den Antenneneingang eines handelsüblichen Fernsehgeräts anschließen zu können, in manchen Taktoszillator-Schaltungen (insbesondere solchen mit Frequenzen oberhalb 10 MHz) und auch in Spannungswandlern, die z.B. zur Gewinnung der für die Programmierung von EPROMs nötigen Spannung von 21 oder 26 V aus der normalen 5-V-Versorgung eingesetzt werden.

Elektrisch gesehen stellen Spulen eine Induktivität dar, deren Wert man in Henry (H) mißt. Ähnlich, wie bei Kondensatoren das Farad eine eigentlich viel zu große Einheit darstellt, bewegen sich die Induktivitäten typischer Spulen im Bereich von Nano- bis Millihenry. Die Wirkung der Spule beruht darauf, daß durch Aufwickeln eines Drahtes das ihn beim Durchfließen eines Stroms unvermeidlich umgebende elektromagnetische Feld auf engem Raum konzentriert wird. In der Tat stellt jede Spule daher auch einen kleinen Elektromagnet dar.

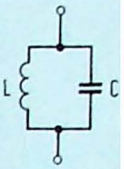
Der Scheinwiderstand einer Spule nimmt im Gegensatz zum Kondensator mit steigender Frequenz zu:

$Z = 2*Pi*L*f$

Pi ist hier wieder die Kreiszahl 3,14..., L die Induktivität in H, f die Frequenz in Hz und Z der Scheinwiderstand der Spule in Ω.

Abb. 1.5.1.4
Schaltung eines Serienschwingkreises mit der Induktivität L und der Kapazität C

Abb. 1.5.1.5
Schaltung eines Parallelschwingkreises



Eine mit einem Kondensator in Serie geschaltete Spule *(Abb. 1.5.1.4)* ergibt einen Serienschwingkreis. Sein Scheinwiderstand Z erreicht bei einer bestimmten Frequenz f ein Minimum:

f = 1/(2*Pi*SQR(L*C))

SQR steht hier für das Wurzelzeichen (siehe das Kapitel über die Programmiersprache Basic!), f ist die Resonanzfrequenz in Hertz und C die Kapazität in F.

Schaltet man Spule und Kondensator nicht in Serie, sondern parallel, hat man einen Parallel-Schwingkreis vor sich. Bei der Resonanzfrequenz (ebenfalls mit obiger Formel zu errechnen) erreicht sein Scheinwiderstand ein Maximum *(Abb. 1.5.1.5)*.

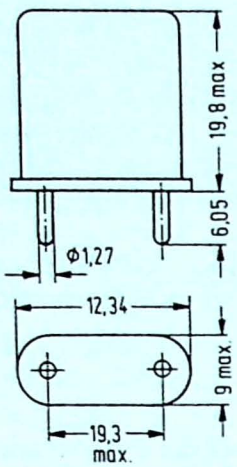
Durch Hineinschieben eines Ferrit- oder Eisenpulver-Kerns in eine Spule erhöht sich deren Induktivität erheblich. Dadurch ist auch eine Justage (ein Abgleich) möglich, ohne erst mühsam die Windungszahl zu ändern. Bei einem sogenannten Schalenkern ist die Spule sogar gänzlich von Ferritmaterial umschlossen, so daß keine magnetischen Feldlinien nach außen treten können.

Wickelt man zwei Spulen aufeinander, so erhält man einen Transformator. Solche „Trafos" findet man vor allem im Netzteil von Computern; sie transformieren die Netzspannung von 220 V auf rund 10 V herunter (Trafos eignen sich wohlgemerkt nur für Wechsel-, nicht für Gleichspannung!). Das Spannungs-Transformationsverhältnis entspricht dem Verhältnis der Windungszahlen auf Primär-(Netz-) und Sekundärseite.

Über die Dimensionierung wird später noch im Zusammenhang mit dem Aufbau von Netzteilen die Rede sein.

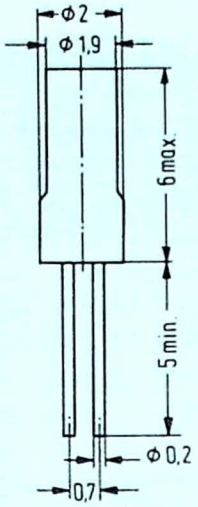
Quarze und Keramik-Resonatoren

Legt man einen entsprechend geschliffenen Quarzkristall oder bestimmte Keramiksorten zwischen zwei Metallplatten, so entspricht ihr elektrisches Verhalten etwa dem eines Schwingkreises. Dabei können sogar mehrere Resonanzstellen auftreten — eine, bei der sich das Material wie ein Serienschwingkreis (niedriger Scheinwiderstand bei Sollfrequenz, Serienresonanz) und eine, bei der er sich wie ein Parallelschwingkreis verhält (hoher Scheinwiderstand bei Sollfrequenz, Parallelresonanz).



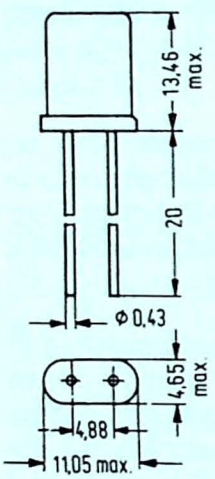
HC-06
HC-33

100 kHz ... 4 MHz



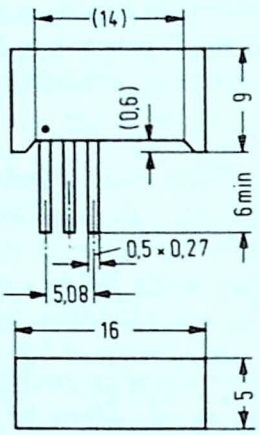
Miniatur
C-002R

Abb. 1.5.1.6
Bauformen von Quarzen und hybriden Quarzoszillatoren



HC-18/U
HC-43/U

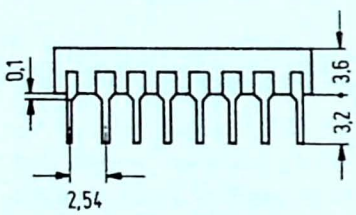
3 MHz ... 70 MHz



SG-10
SG-11

10 Hz ... 16 MHz

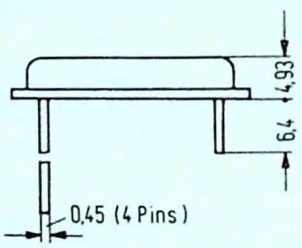
CMOS-Quarzoszillatoren mit integriertem Quarz



SPG-Serie

0,0005 Hz ... 1 MHz

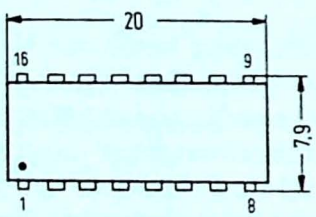
Programmierbare Quarzoszillatoren mit integriertem Quarz

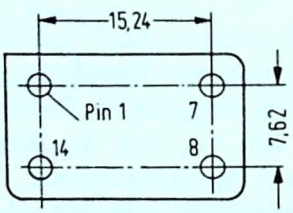


QOS

100 kHz ... 70 MHz

Hybride Quarzoszillatoren





Maße in mm

Gegenüber einem Schwingkreis besitzen solche mechanisch schwingenden Resonatoren (sie biegen sich im Takt der angelegten Wechselspannung) wesentliche Vorteile: Die Frequenz ist sehr stabil und auch weitgehend unabhängig von äußeren Einflüssen wie Temperatur und externen Schaltungskapazitäten (z.B. der Eigenkapazität der Leiterbahnen einer Platine). Ein Abgleich ist daher meist gar nicht nötig.

Quarze gibt es im Frequenzbereich von etwa 10 kHz bis 100 MHz. *Abb. 1.5.1.6* zeigt Gehäuseformen von handelsüblichen Quarzen. Die Typgruppen SG, SPG und QOS sind dabei komplette integrierte Quarzoszillatoren. Die nicht ganz so hochwertigen Keramik-Resonatoren findet man dagegen im Bereich von 100 kHz bis 30 MHz. In Computern werden Quarze und Keramikschwinger vorzugsweise im Taktoszillator eingesetzt.

Halbleiter

Ein Material, dessen Leitfähigkeit für elektrischen Strom zwischen der von Metallen (Leitern) und Isolatoren (Nichtleitern) einzuordnen ist, bezeichnet man als Halbleiter. Dazu gehören u.a. die Elemente Germanium und Silizium sowie Gallium-Arsen-Verbindungen. Davon spielt in der Computertechnik allerdings nur Silizium eine namhafte Rolle.

Absolut reines Silizium leitet den elektrischen Strom überhaupt nicht. Erst durch gezielte Verunreinigung des Silizium-Kristalls mit Fremdelementen, allerdings nur im Promille-Bereich, kann man die gewünschten Eigenschaften erzielen. Indem man einen kleinen Silizium-Kristall nicht gleichmäßig, sondern nur an bestimmten Teilen und mit unterschiedlichen Elementen und Konzentrationen verunreinigt („dotiert"), kann man ihn sogar dazu bringen, Strom nur in einer Richtung zu leiten. Bei solchen zweipoligen Halbleiter-Bauelementen mit Gleichrichter-Wirkung spricht man von Dioden.

Gleichrichter-Dioden

Die Eigenschaften von Halbleiter-Bauelementen kann man am besten grafisch umschreiben, bei Dioden z.B. durch eine „Kennlinie" *(Abb. 1.5.1.7)*, die dadurch entsteht, daß man in x-Richtung die angelegte Spannung und in y-Richtung den dann fließenden Strom aufzeichnet. (Bei einem ohmschen Widerstand ergäbe sich dabei eine Gerade durch den Nullpunkt, deren Steigung durch den Widerstandswert bestimmt wird.)

Der Kennlinie läßt sich entnehmen, daß in der einen Richtung (der Durchlaßrichtung) erst ab einer bestimmten „Schwellspannung" ein Strom fließt, nämlich von der Katode zur Anode (der aus Elektronen bestehende elektrische Strom fließt vom Minus- zum Pluspol). Diese Schwellspannung liegt bei Silizium-Dioden um 0,6 V, bei Germanium-Dioden um 0,2 V und bei Schottky-Dioden (sie arbeiten mit einem Übergang von Silizium auf Metall) um 0,4 V.

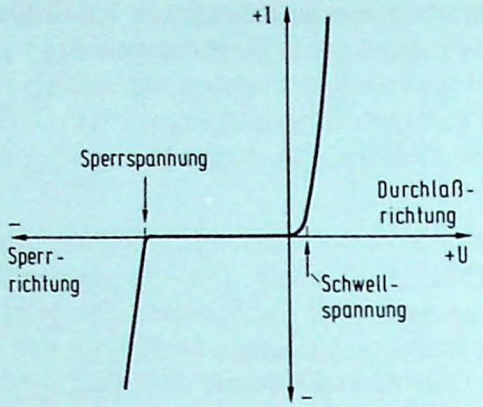


Abb. 1.5.1.7
Kennlinie einer Z- oder Zener-Diode

In der anderen Richtung, der Sperrichtung, fließt ebenfalls ab einer bestimmten Spannung ein Strom, nämlich ab der Sperrspannung. Sie ist allerdings sehr starken Fertigungstoleranzen unterworfen. Bei Gleichrichterdioden kann sie 20...2000 V betragen. Sie sollte im Normalbetrieb auf keinen Fall erreicht werden, da das zu einer Beschädigung führen könnte.

Z-Dioden

Das fast schlagartige Einsetzen eines Stromflusses ab einer bestimmten Spannung in Sperrichtung kann auch genutzt werden, beispielsweise zur Spannungs-Stabilisierung in Stromversorgungs-Schaltungen *(Abb. 1.5.1.8)*. Fertigungstechnisch wird dafür gesorgt, daß die Sperrspannung dieser sogenannten Zener- oder Z-Dioden den gewünschten Wert mit geringer Toleranz einhält. Die Spannung vor dem der Strombegrenzung dienenden Widerstand muß natürlich höher als die Z-Diodenspannung sein, die gleich der gewünschten Ausgangsspannung ist.

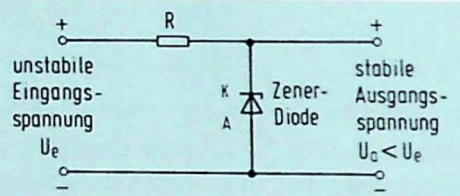


Abb. 1.5.1.8
Prinzip der Stabilisation einer Spannung mit Hilfe einer Z-Diode

LEDs

Leuchtdioden, kurz LEDs (Light Emitting Diodes), werden nicht aus Silizium, sondern aus Galliumarsenid oder Galliumphosphid hergestellt. Sie besitzen eine Schwellspannung von 1,5 V (rotleuchtend) oder rund 2 V (gelb und grün). Blaue Leuchtdioden lassen sich aus Karbid-Verbindungen herstellen; da die erreichbaren Kristallgrößen dabei aber recht klein sind, sind blaue Leuchtdioden schwierig zu fertigen und deshalb auch kaum zu finden.

Leuchtdioden werden gründsätzlich in Durchlaßrichtung und mit Strömen von etwa 5...50 mA betrieben. Man kann aus mehreren solcher Dioden in einem gemeinsamen Gehäuse auch z.B. Siebensegment-Anzeigen zur Darstellung von Ziffern realisieren.

Flüssigkristalle

Da wir gerade von Anzeigen sprechen: Wesentlich stromsparender als LEDs sind LCDs (Liquid Crystal Displays). Sie bestehen aus zwei metallbedampften Glasplatten, zwischen denen sich eine organische Flüssigkeit befindet — also nur im weitesten Sinn ein Halbleiter. Ihre Kristalle drehen sich je nach angelegter Spannung längs oder quer zur Blickrichtung des Betrachters und verursachen so eine scheinbare Verfärbung.

LCDs werden für die Darstellung von Ziffern aus sieben, für Buchstaben aus 12 oder 16 Segmenten pro Zeichen zusammengesetzt. Oft sind die aufgedampften Metallflächen auch als Punktraster ausgeführt, z.B. bei tragbaren Computern als Ersatz für die altertümliche Bildröhre.

Die zum Betrieb einer Flüssigkristall-Anzeige nötige Spannung beträgt etwa 1,5...20 V, der aufgenommene Strom nur einige Mikroampere pro Element (das kann sich bei einer Punktmatrix-Anzeige allerdings schnell auf einige mA summieren).

Eine Besonderheit ist, daß LCDs nicht mit Gleichspannung, sondern mit einer Spannung alternierender Polarität betrieben werden. Andernfalls würde sich die Flüssigkeit elektrolytisch zersetzen. Dieser Tatsache muß man durch besondere Steuerschaltungen Rechnung tragen.

Bipolare Transistoren

Halbleiter lassen sich auch als Verstärker einsetzen. Man kann sich das so vorstellen: Nimmt man zwei Dioden und preßt sie mit ihren Katoden eng aufeinander, so erhält man einen „bipolaren" Transistor (zur Unterscheidung von Feldeffekt-Transistoren, s.u.). Die Ladungsträger der einen Diodenstrecke (Halbleiter-Sperrschicht) können die Leitfähigkeit der anderen beeinflussen *(Abb. 1.5.1.9)*. Setzt man für Anode (Pluspol) den Buchstaben p und für Katode (negativer Pol) ein n, so ergibt sich eine npn-Struktur dieses Halbleiter-Bauelements. Es ist aber auch möglich, die beiden Dioden mit ihrer Anodenseite aufeinanderzupressen; das Resultat wäre ein pnp-Bauelement.

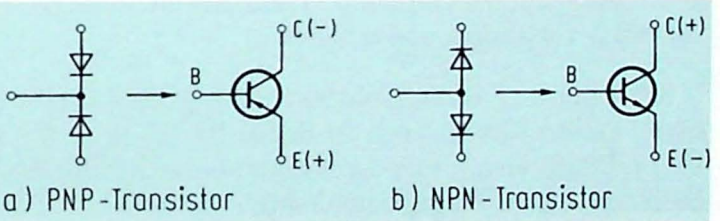


Abb. 1.5.1.9
Bipolare Transistoren bestehen aus zwei „aufeinandergepreßten" Diodenstrecken

Ein Transistor besitzt die drei Anschlüsse Basis (Steuerelektrode), Emitter (bei npn-Transistoren der Minuspol, er emittiert also Elektronen) und Collector („Aufsammler" für Elektronen und damit Pluspol). Solange kein Strom in die Basis fließt, sperrt der Transistor, d.h. er läßt keinen Strom vom Emitter zum Collector durch. Aber schon ein sehr geringer Strom in die Basis erlaubt es, einen sehr viel größeren Strom vom Emitter zum Collector fließen zu lassen. Das Verhältnis von Basis- zu Collectorstrom nennt man Stromverstärkung.

In Computern werden Transistoren überwiegend als elektronische Schalter eingesetzt: Mit einem geringen Basis-Steuerstrom kann man einen größeren Collectorstrom steuern und damit schalten. Die Basis-Emitter-Strecke wirkt dabei wie eine Diodenstrecke. Der Collectorstrom kann erst fließen, wenn die Basisspannung die Schwellspannung von etwa 0,6 V überschreitet und damit ein Strom in die Basis fließt.

Transistoren können je nach Typ Ströme von etwa 50 mA bis zu einigen Ampere und Spannungen von 20...1000 V schalten und tun das klaglos auch einige Millionen Mal pro Sekunde.

Feldeffekt-Transistoren

Bipolare Transistoren beruhen auf einer Stromsteuerung. Feldeffekt-Transistoren sind grundsätzlich anders aufgebaut. Sie bestehen aus einem dünnen „Kanal", durch den der zu steuernde Strom fließt, und einer vom Kanal isolierten Steuerelektrode (Gate). Durch die Einwirkung des elektrischen Feldes, das je nach angelegter Steuerspannung vom Gate auf den Kanal ausgeht, ändert sich das Durchlaßverhalten des Kanals; seine beiden Anschlüsse heißen Source (Minuspol = Quelle der Elektronen) und Drain (Pluspol = Senke).

Die Isolation der Steuerelektrode vom Kanal kann durch eine Dioden-Sperrschicht (Sperrschicht-FET) oder eine dünne Metall-Oxid-Schicht erfolgen (MOSFET = Metall-Oxid-Silizium-FET). MOSFETs kann man auch mit zwei Gate-Elektroden herstellen; sie heißen dann Dual-Gate-MOSFETs.

Wichtigstes Merkmal der Feldeffekt-Transistoren ist, daß die Steuerelektrode keinen Strom aufnimmt, da sie ja vom Kanal isoliert ist. Man kann daher nicht von einer Strom-, sondern nur von einer Spannungs-Verstärkung sprechen, die je nach Schaltungsauslegung etwa den Faktor 5...30 erreichen kann. Der zu steuernde Strom, der durch den Source-Drain-Kanal fließt, darf meist 20...100 mA nicht überschreiten. Wie bipolare Transistoren können FETs Frequenzen bis über 100 MHz verstärken und schalten.

Schließlich ist es auch möglich, FETs auch so herzustellen, daß die Source-Elektrode nicht als Minuspol (n-Kanal-FET), sondern als Pluspol dient (p-Kanal-FET). Damit lassen sich die sogenannten CMOS-ICs realisieren, auf die im übernächsten Kapitel eingegangen wird.

Weitere Halbleiter-Bauelemente

Neben den verschiedenen Dioden- und Transistortypen gibt es noch weitere Halbleiter-Bauelemente, die in Computer-Schaltungen allerdings seltener anzutreffen sind und daher hier nicht ausführlich besprochen werden.

Während Transistoren aus zwei Sperrschichten aufgebaut sind, bestehen Thyristoren aus drei. Sie können hohe Spannungen und hohe Ströme schalten, kennen aber nur die Zustände „ein" und „aus". Triacs stellen Thyristoren dar, die auch Wechselstrom schalten können (bis zu einigen zehn kHz).

Schließlich sind noch die unterschiedlichen Halbleiter-Sensoren zu erwähnen, die in einem eigenen Kapitel besprochen werden.

1.5.2 TTL-Schaltungen

Um auf die Bauelemente-Anzahl und damit den Preis elektronischer Geräte zu reduzieren, kommen heute integrierte Schaltungen (ICs = Integrated Circuits) zum Einsatz. Die älteste „IC-Familie" ist die der TTL-Bausteine. TTL bedeutet Transistor-Transistor-Logik, also eine Technik, bei der in einem IC mehrere Transistoren jeweils zu einem Logik-Baustein zusammengefaßt sind.

Etwas Logik

Arithmetische und logische Schaltungen, auch komplette Mikroprozessoren oder Speicherbausteine, sind aus einzelnen Logik-Grundschaltungen aufgebaut, aus sogenannten Gattern. Das sind Schaltungen, die aus einem oder mehreren Eingangs-Signalen ein bestimmtes Ausgangssignal machen. Der Einfachheit halber läßt man nur zwei Signalpegel an Ein- und Ausgängen zu, die man mit 0 und 1 bezeichnet. 0 sei eine Spannung von 0...0,8 V, 1 entspreche 2,4...5 V (siehe Kapitel über Mikroprozessoren). Alle anderen Spannungen sind verboten und nicht definiert. 1 und 0 kann man natürlich auch als Ziffern des dualen Zahlensystems auffassen, und das ist sogar sehr sinnvoll. Die Rechenkunst mit 0 und 1 nennt man Boole'sche Algebra (nach einem gewissen Boole, der das längst vor dem ersten Computer erfunden hat).

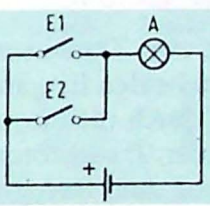


Abb. 1.5.2.1 Oder-Verknüpfung mit zwei Schaltern. Die Lampe A leuchtet, wenn E1 oder E2 oder beide Schalter geschlossen sind

In dieser Primitiv-Algebra mit nur zwei Ziffern gibt es folgende Verknüpfungen:

Funktionen, Symbol	Eingangs-signal(e) E1	E2	Ausgangs-signal A
Inverter (Nicht) $A=\overline{E1}$	0		1
	1		0
AND (Und) $A=E1 \wedge E2$	0	0	0
	0	1	0
	1	0	0
	1	1	1
OR (Oder) $A=E1 \vee E2$	0	0	0
	0	1	1
	1	0	1
	1	1	1
NAND (Nicht-Und) $A=\overline{E1 \wedge E2}$	0	0	1
	0	1	1
	1	0	1
	1	1	0
NOR (Nicht-Oder) $A=\overline{E1 \vee E2}$	0	0	1
	0	1	0
	1	0	0
	1	1	0
EXOR (Exklusiv-Oder) $A=E1 \forall E2$	0	0	0
	0	1	1
	1	0	1
	1	1	0

Diese logischen Funktionen lassen sich ohne weiteres elektrisch nachbilden. *Abb. 1.5.2.1* zeigt eine Oder-Verknüpfung mit zwei Schaltern; 1 entspricht einem geschlossenen Schalter bzw. einer leuchtenden Lampe. Die Lampe leuchtet, wenn einer der beiden Schalter oder beide geschlossen sind.

Abb. 1.5.2.2 enthält eine mit den gleichen Bauelementen realisierte Und-Verknüpfung. Hier leuchtet die Lampe nur dann, wenn beide Schalter geschlossen sind. Natürlich läßt sich dieses Prinzip auch auf mehr Schalter (respektive Logik-Gatter-Eingänge) übertragen.

Wie man mit zwei Transistoren ein NAND-Gatter realisieren kann, zeigt *Abb. 1.5.2.3.* Wenn an beiden Eingängen eine Spannung von mehr als etwa 1 V liegt, also log. 1, fließt durch die Strombegrenzungs-Widerstände ein Strom in die Basis-Anschlüsse beider Transistoren, so daß deren Collector-Emitter-Strecken leiten und den Ausgang zum Bezugspotential (Masse, 0 V) kurzschließen. Andernfalls

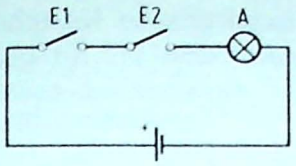


Abb. 1.5.2.2 Und-Verknüpfung: A leuchtet, wenn E1 und E2 gleichzeitig geschlossen sind

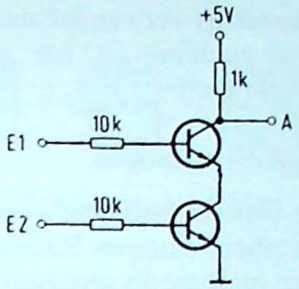


Abb. 1.5.2.3 NAND-Gatter mit zwei Transistoren. Der Ausgang A geht auf Low-Pegel, wenn E1 und E2 auf High-Pegel liegen

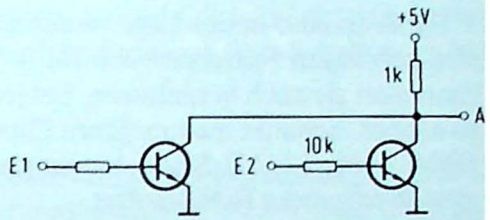


Abb. 1.5.2.4 NOR-Gatter mit zwei Transistoren. A geht auf Low, wenn E1 oder E2 oder beide Eingänge auf High-Pegel liegen

wird der Ausgang durch den Widerstand am Collector des oberen Transistors auf 5 V, also log. 1, gehalten.

Ein NOR-Gatter mit zwei Transistoren ist in *Abb. 1.5.2.4* gezeichnet. Hier genügt es schon, wenn einer der beiden Transistoren leitet, wenn also wenigstens an einem Eingang log. 1 liegt, um den Ausgang nach Masse kurzzuschließen, was log. 0 entspricht.

Logik-Symbole

Es wäre ziemlich umständlich, in einer umfangreichen Logikschaltung jedes Gatter so zu zeichnen, wie es tatsächlich aufgebaut ist. Man hat daher Symbole eingeführt, die logische Verknüpfungen übersichtlich darstellen. Leider gibt es dafür allerdings keine internationale Norm. Erst vor wenigen Jahren wurde ein DIN-Standard geschaffen, der praktisch nur hierzulande verwendet wird. *Abb. 1.5.2.5* zeigt die unterschiedlichen Darstellungsweisen für Logik-Gatter.

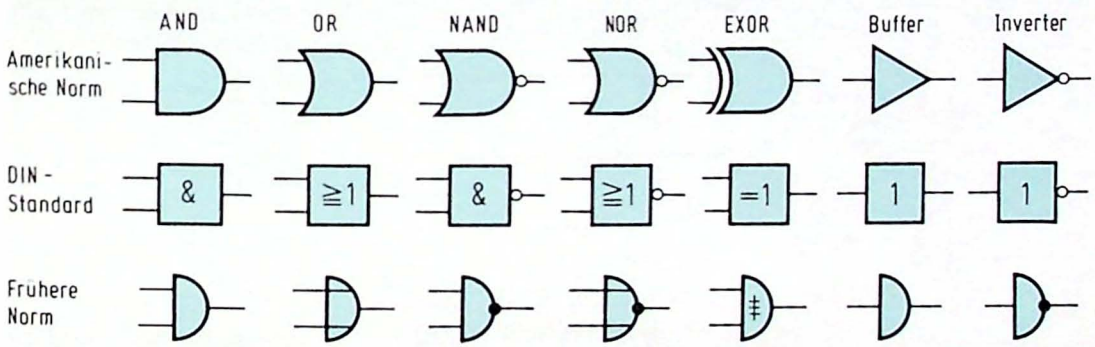


Abb. 1.5.2.5 Symbole für Logik-Gatter nach unterschiedlichen Normen

Der DIN-Standard verwendet dabei prinzipiell quadratische Kästchen, wobei die Funktion oft auch mit „&" für „und", „+" für „oder" und „—" für Invertierung (Nicht) dargestellt wird.

Komplexe Logik-Schaltungen

Aus einzelnen Gattern lassen sich auch komplexere Schaltungen zusammenfassen, die oftmals in einem einzigen Bauelement integriert sind. Dazu gehören Decoder (sie verknüpfen mehrere Eingangssignale zu einem oder mehreren Ausgangssignalen, z.B. Adressen-Decoder), Flipflops *(Abb. 1.5.2.6)* und die aus Flipflops zusammengesetzten Schieberegister und Zähler.

Flipflops sind in der Lage, sich bestimmte logische Zustände zu „merken", sind also sozusagen Speicherzellen für 1 Bit. Durch geeignete Schaltungsmaßnahmen kann man sie auch veranlassen, bei jedem an ihren Takteingang angelegten Impuls zu kippen. Schaltet man mehrere Flipflops hintereinander, erhält man einen Binärzähler *(Abb. 1.5.2.7)*. Seine Ausgangssignale wechseln nach jedem Taktimpuls, und zwar in folgender Reihenfolge:

Taktimpulse	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Q0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Q1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
Q2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0
Q3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0

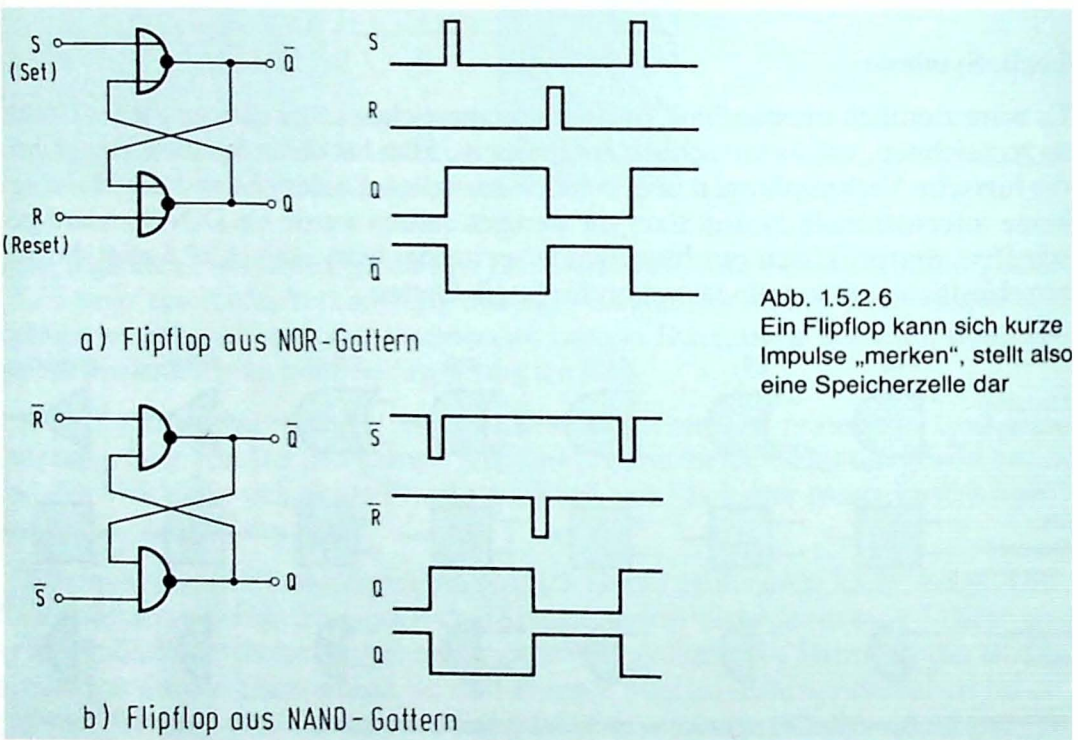


Abb. 1.5.2.6
Ein Flipflop kann sich kurze Impulse „merken", stellt also eine Speicherzelle dar

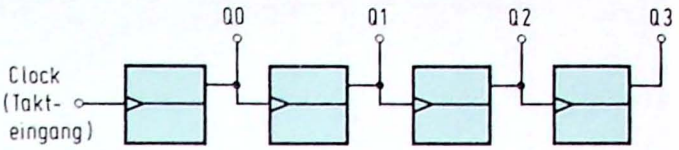


Abb. 1.5.2.7
Mit vier Flipflops kann man einen 4-Bit-Binärzähler realisieren

Es ist leicht zu erkennen, daß die Zahl der Taktimpulse an den Ausgängen als Dualzahl zur Verfügung steht.

Solche Zähler verfügen meist zusätzlich über einen Rückstelleingang, mit dem sich alle Ausgänge auf Null stellen lassen. Damit ist es auch möglich, einen Dezimalzähler zu realisieren: Man braucht ja nur mit einer Und-Verknüpfung dafür zu sorgen, daß die Rückstellung automatisch erfolgt, sobald Q3 und Q2 auf 1 gehen. Wenden wir uns nun aber von grauer Theorie ab und realen Logik-Schaltungen zu!

TTL-Schaltungen

Die ICs der „74-Familie" eignen sich für eine Versorgungsspannung von 5 V (+/− 5%). Log. 0 (Low-Pegel) ist der Spannungsbereich 0...0,8 V, Log. 1 (High-Pegel) entspricht 2,4...5 V. Ein IC enthält gewöhnlich mehrere Gatter oder eine recht komplexe Logikschaltung. Die meisten TTL-ICs besitzen ein 14- oder 16poliges Dual-In-Line-Gehäuse (DIL) und bewegen sich im Preisbereich 1...5 DM.

Abb. 1.5.2.8 zeigt die Innenschaltung eines der vier NAND-Gatter im IC 7400. Es gibt auch eine Version 74L00 mit geringem Stromverbrauch, aber auch geringer Schaltgeschwindigkeit *(Abb. 1.5.2.9)*, den Typ 74S00 in Schottky-Technik mit sehr hoher Schaltgeschwindigkeit *(Abb. 1.5.2.10)* und schließlich die Version 74LS00, bei der man etwa die gleiche Geschwindigkeit wie beim 7400 mit dem geringen Stromverbrauch des 74L00 kombiniert hat.

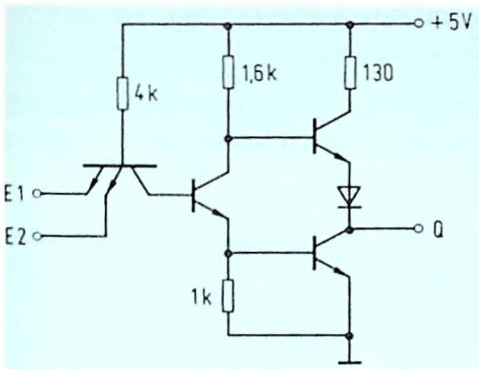


Abb. 1.5.2.8 Innenschaltung eines Standard-TTL-NAND-Gatters der 74-Serie

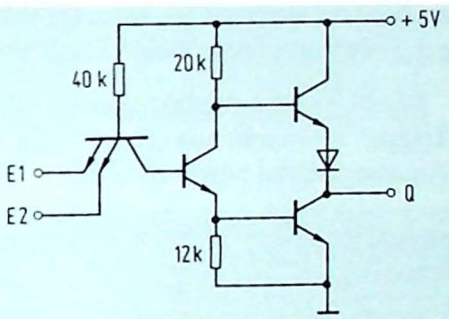


Abb. 1.5.2.9 Um einen niedrigeren Stromverbrauch zu erzielen, wurden bei der 74-L-Serie die Widerstände größer gewählt

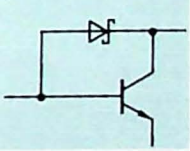


Abb. 1.5.2.10 Bei Schottky-TTL-Schaltungen (74 S, LS) verhindert eine Schottky-Diode niedriger Schwellspannung, daß der Schalttransistor übersteuert (gesättigt) wird und langsamer schaltet

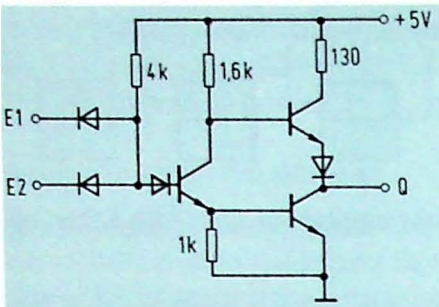


Abb. 1.5.2.11 Den Multiemitter-Transistor am NAND-Gatter-Eingang kann man sich als drei Dioden vorstellen

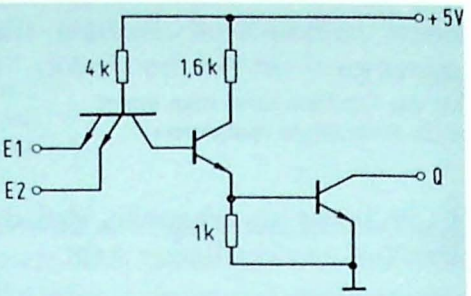


Abb. 1.5.2.12 Schaltung eines Standard-TTL-NAND-Gatters mit offenem Kollektor am Ausgang

Die beiden Eingänge sind mit einem Multi-Emitter-Transistor verbunden, den man sich aber in Wahrheit als Diodenstrecken-Kombination vorzustellen hat *(Abb. 1.5.2.11)*. Nur wenn beide Eingänge auf High-Pegel liegen, wenn also keiner der beiden das untere Ende des Basiswiderstands nach Masse kurzschließt, kann auch der Collector auf High-Pegel gehen und steuert den nachfolgenden Inverter an, so daß dann der untere Ausgangstransistor leitet und der obere sperrt (Low-Pegel am Ausgang).

Es gibt auch TTL-ICs, bei denen am Ausgang nur ein einziger Transistor mit „offenem Collector" (open collector) vorhanden ist *(Abb. 1.5.2.12)*. Damit ist es auch möglich, am Ausgang Spannungen von mehr als 5 V zu schalten (beim 7406 z.B. etwa 20 V. Solange ein solcher Ausgang allerdings unbeschaltet ist, kann man an ihm natürlich keine Spannung messen.

Unbeschaltete TTL-Eingänge liegen, bedingt durch die schon geschilderte Multiemitter-Schaltung auf log. 1 (also High-Pegel). Man kann also ein NAND-Gatter auch ohne weiteres als Inverter verwenden, indem man entweder nur einen Eingang beschaltet oder beide Eingänge miteinander verbindet.

Nicht so ganz digital arbeitet, zumindest eingangsseitig, der sogenannte Schmitt-Trigger: Er macht aus einem stetig veränderlichen Eingangssignal ein rein digitales Ausgangssignal *(Abb. 1.5.2.13)*.

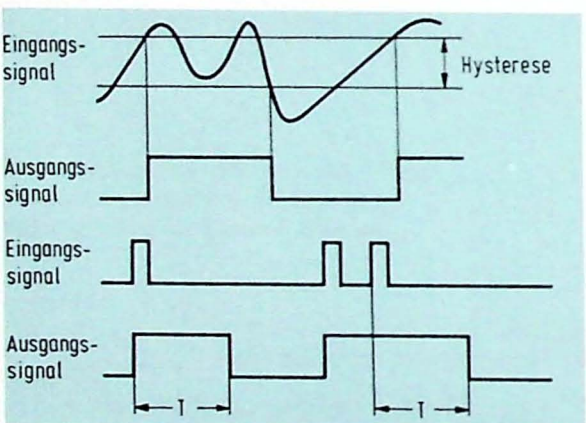
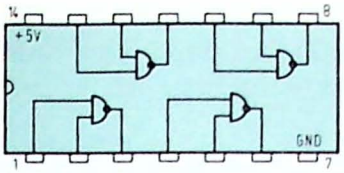
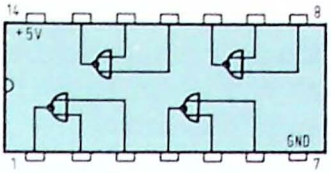


Abb. 1.5.2.13
Ein Schmitt-Trigger macht aus einem analogen ein digitales Signal

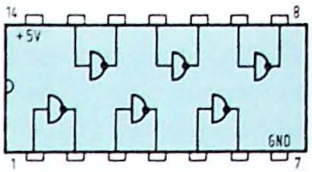
Abb. 1.5.2.14 Mit einem Monoflop kann man Impulse bestimmter Zeitdauer erzeugen



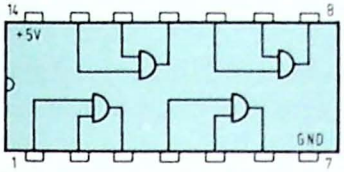
7400 (TTL-Ausgang),
7403 (off Collector) 4×NAND



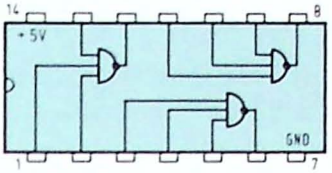
7402 4×NOR



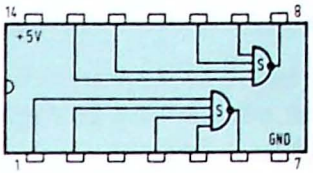
7404 (TTL-Ausgang),
7406 (off Collector) 6 Inverter



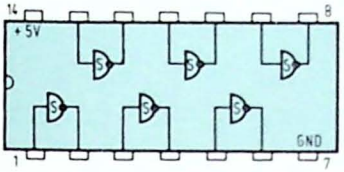
7408 4×AND



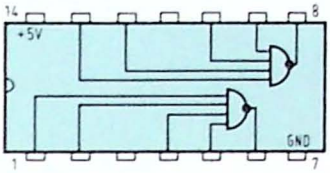
7410 3×NAND



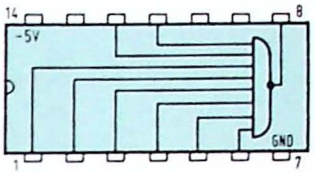
7413 2 Schmitt-Trigger



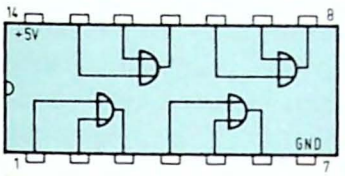
7414 6 Schmitt-Trigger



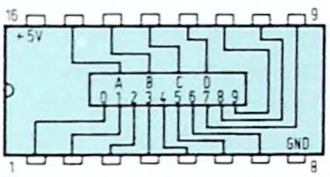
7420 (TTL-Ausgang),
7422 (off Collector) 2×NAND



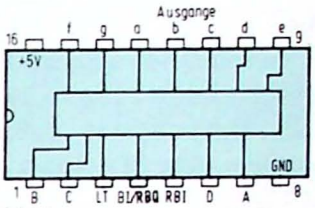
7430 1×NAND



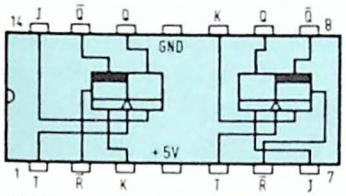
7432 4×OR



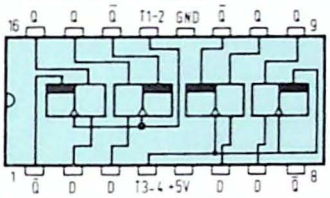
7442 BCD-Dezimal-Decoder
74145 mit off. Collector



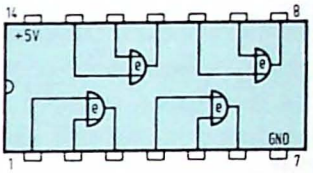
7447 BCD-Siebensegment-Decoder



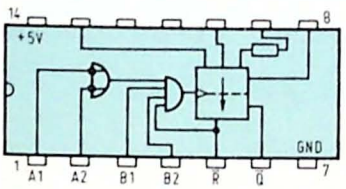
7473 2 JK-Flipflops



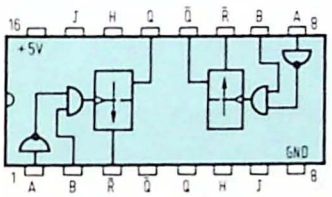
7475 4D-Flipflops



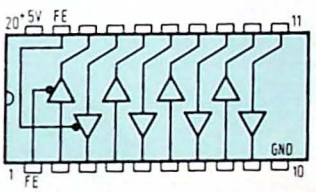
7486 4×EXOR



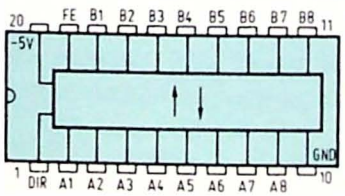
74122 Nachtriggerbares
Monoflop



74123 Nachtriggerbare
Monoflops



74244 Abschaltbare Buffer
(FE = Low → Ein)



74245
Bidirektionaler 8-Bit-Buffer (invertierend: 74645), abschaltbar mit FE = High, DIR = Low; Richtung B → A; High: A → B

Abb. 1.5.2.15
Anschlußbilder häufig verwendeter TTL-Schaltungen (von der Beschriftungsseite gesehen)

Monoflops dienen zur Erzeugung von Impulsen, deren Dauer durch eine Kombination aus einem Widerstand und einem Kondensator vorgebbar ist: Je nach Schaltung gilt näherungsweise dabei die Formel

$T = R * C$

mit T = erzeugte Impulsdauer in s, C = Kondensator in F, R = Widerstand in Ω. Beispiel: Mit C = 1 µF und R = 1 MΩ ergibt sich T = 1 s.

Ein Monoflop bezeichnet man als „nachtriggerbar", wenn ein Eingangsimpuls, der während des erzeugten Impulses T auftritt, diesen entsprechend verlängert *Abb. 1.5.2.14)*.

Die Anschlußbilder der üblichsten TTL-ICs gehen aus *Abb. 1.5.2.15* hervor.

1.5.3 CMOS-Schaltungen

Die TTL-Technik eignet sich verhältnismäßig schlecht für batteriebetriebene Geräte, da schon ein einziges Gatter einige mA Strom verbraucht und z.B. ein Adressendecoder in einem Computer schon aus einer Vielzahl von Gattern bestehen kann. Hinzu kommt, daß dadurch auch bei netzbetriebenen Geräten ein Wärmeproblem auftreten kann, denn die entstehende Verlustleistung (Speisespannung mal Strom ergibt Leistung!) muß ja irgendwie abgeführt werden.

Die CMOS-Technik verwendet MOS-Feldeffekt-Transistoren *(Abb. 1.5.3.1)*. Wie schon erwähnt, ist ihre Steuerelektrode, das Gate, vom Drain-Source-Kanal vollständig isoliert und benötigt keinen Strom. Damit auch der Kanal keinen Stromfluß verursacht, schaltet man einfach einen p- und einen n-Kanal-FET in Reihe, von denen immer nur einer gerade leitet. *Abb. 1.5.3.2* zeigt den so entstandenen CMOS-Inverter (Complementary MOS = Komplementär-MOS).

Liegt der Eingang E auf High-Pegel, so wird der untere n-Kanal-MOSFET leitend und der obere p-Kanal-MOSFET sperrt. Der Ausgang Q führt dann also

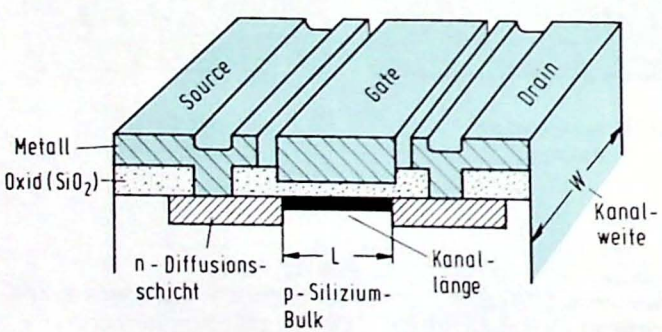


Abb. 1.5.3.1
Aufbau eines Metalloxid-Feldeffekt-Transistors (MOSFET)

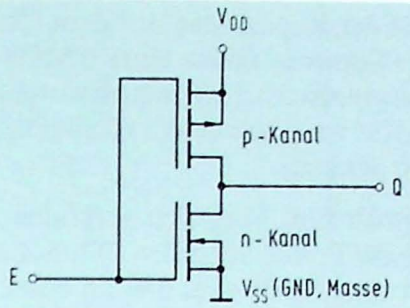


Abb. 1.5.3.2
Schaltung eines CMOS-Inverters

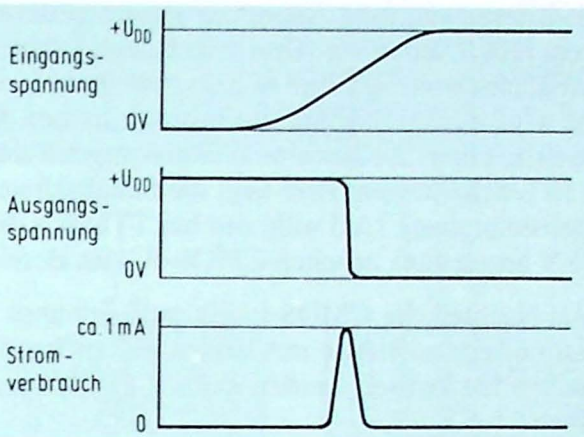


Abb. 1.5.3.3
Nur im Umschaltmoment fließt bei einem CMOS-Gatter kurzzeitig ein Strom zum Umladen interner Kapazitäten

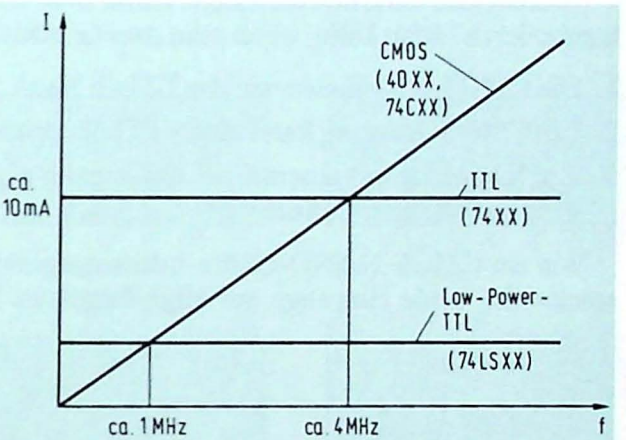


Abb. 1.5.3.4
Vergleich der Leistungsaufnahme von TTL- und CMOS-Logik je nach Arbeitsfrequenz

Low-Pegel. Liegt E auf Low, so sperrt der untere FET und der obere leitet, so daß Q High-Pegel führt.

Strom fließt nur während der Umschaltzeit *(Abb. 1.5.3.3)*, nämlich erstens, weil in einem bestimmten Eingangsspannungsbereich beide FETs leiten können (das hängt von der Halbleiter-Dotierung und der angelegten Versorgungsspannung ab) und zweitens, weil ja die unvermeidlichen internen und externen Kapazitäten umgeladen werden müssen.

Dieses Umladen der Kapazitäten auf dem Chip und der externen Verdrahtung führt dazu, daß die Stromaufnahme eines CMOS-Gatters in guter Näherung direkt proportional der angelegten Schaltfrequenz ist *(Abb. 1.5.3.4)*. Dabei entspricht eine Frequenz von 1 MHz natürlich in Wirklichkeit zwei Millionen Schaltvorgängen in jeder Richtung pro Sekunde.

Abb. 1.5.3.4 enthält zum Vergleich auch den Stromverbrauch von herkömmlichen TTL- und Low-Power-Schottky-TTL-Schaltungen, der praktisch frequenzunabhängig ist. Daraus ergibt sich, daß ab einer bestimmten Frequenz, die in der Grafik als Schnittpunkt sichtbar ist, CMOS-Gatter tatsächlich mehr Strom als TTL-ICs verbrauchen! Allerdings kommen derart häufige Schaltzyklen bei Computern relativ selten vor. Und selbst dann können CMOS-ICs mit einem weiteren Vorteil aufwarten: Da ihre Schaltschwelle bei etwa der halben Versorgungsspannung liegt, dürfen Störimpulse fast den halben Wert dieser Versorgungsspannung erreichen, ohne daß dies eine Auswirkung auf die korrekte Funktion hätte. Anders bei TTL-Schaltungen: Hier liegt die Schaltschwelle bei etwa 1,6 V (bei 5 V Spannungsversorgung). Und während bei TTL-ICs die Spannung zwischen 4,75 V und 5,25 V liegen muß, arbeiten CMOS-ICs im Bereich 3...15 V ohne Probleme.

Als Nachteil der CMOS-Logik muß erwähnt werden, daß ein CMOS-Ausgang meist nur einige wenige mA Strom liefern kann, was schon beim Anschluß einer Leuchtdiode kritisch werden kann. TTL-Ausgänge können dagegen 10...40 mA treiben.

Da stellt sich natürlich die Frage: Kann man CMOS-Schaltungen mit TTL-Logik kombinieren? Man kann, wenn man drei Grundsätze beachtet:

1. Die CMOS-ICs müssen wie die TTL-ICs mit 5 V betrieben werden.
2. Ein CMOS-Ausgang kann einen TTL-Eingang direkt treiben.
3. Zur Kopplung mit einem CMOS-Eingang eignet sich am besten ein TTL-Ausgang mit offenem Collector *(Abb. 1.5.3.5)*.

Wie ein CMOS-NAND-Gatter intern aufgebaut ist, zeigt *Abb. 1.5.3.6.* Sind beispielsweise beide Eingänge auf High-Pegel, so leiten die beiden unteren in Serie

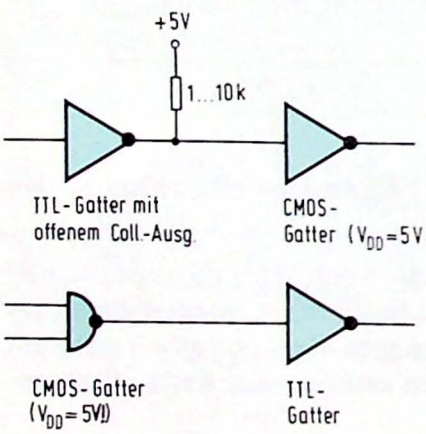


Abb. 1.5.3.5
Die Verbindung von TTL- und CMOS-Logik ist bei gleicher Betriebsspannung weitgehend problemlos

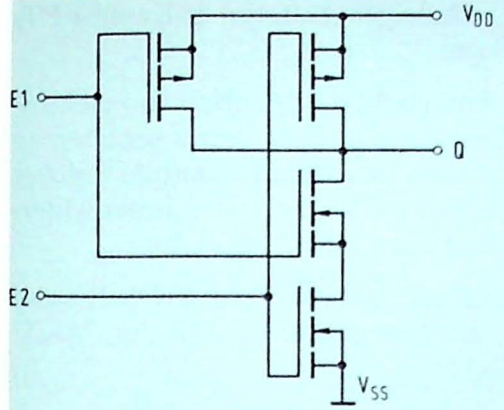


Abb. 1.5.3.6
NAND-Gatter in CMOS-Technik

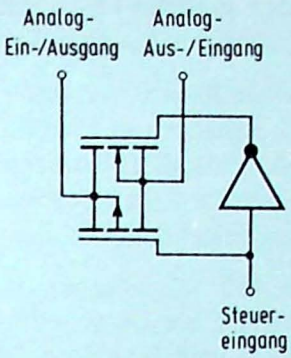
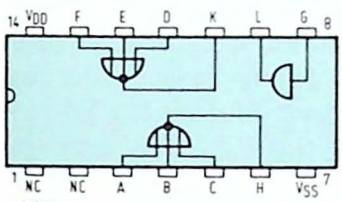
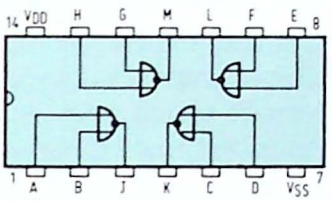


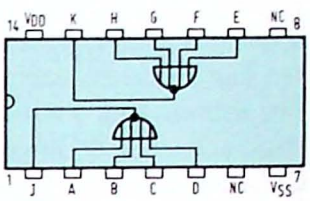
Abb. 1.5.3.7 Elektronischer Schalter in CMOS-Technik. Damit ist es möglich, auch analoge Signale zu schalten



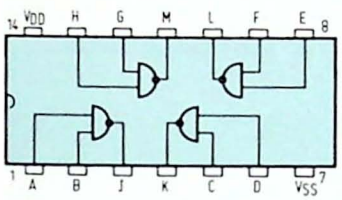
4000 1×Buffer, 2×NOR



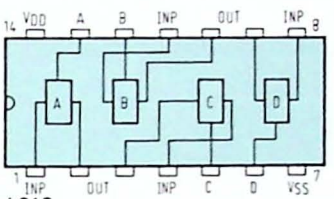
4001 4×NOR



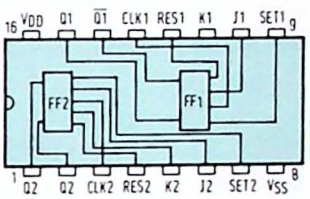
4002 2×NOR



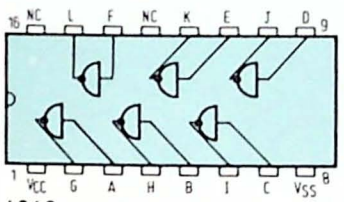
4011 4×NAND



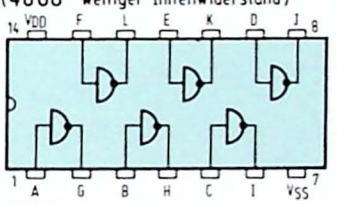
4016 4 Analogschalter
(4066 weniger Innenwiderstand)



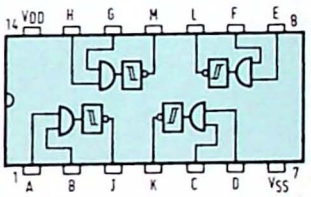
4027 2 Flipflops



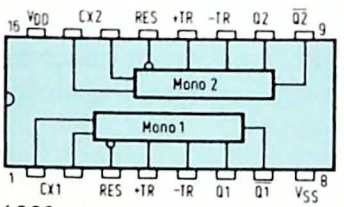
4049 6 Inverter



4069 6 Inverter



4093 4 NAND-Schmitt-Trigger



4098 2 Monoflops

Abb. 1.5.3.8 Anschlußbilder einiger CMOS-ICs der 4000-Serie

liegenden n-Kanal-FETs, die beiden oberen parallelgeschalteten p-Kanal-FETs sperren und der Ausgang liegt daher auf Low-Pegel.

Das sehr lineare Verhalten der FET-Kanäle ermöglicht den Aufbau von elektronischen Schaltern in CMOS-Technik, um auch analoge Signale damit schalten zu können, ohne diese zu verzerren. *Abb. 1.5.3.7* zeigt das Innenschaltbild solcher Analogschalter. Ihr Sperrwiderstand ist fast unmeßbar hoch — der Innenwiderstand im eingeschalteten Zustand beträgt allerdings etwa 100...300 Ohm.

Aus *Abb. 1.5.3.8* gehen die Anschlußbilder einiger verbreiteter CMOS-ICs der Serie 40XX hervor. Es gibt auch CMOS-ICs, die pinkompatibel mit der 74XX-TTL-Serie sind; sie heißen 74CXX oder 74XCXX.

1.5.4 Stromversorgungs-Schaltungen

Die meisten Mikroprozessoren, Speicher und Peripheriebausteine, aber auch TTL-ICs, verlangen eine stabile Versorgungsspannung, die maximal um 5% vom Sollwert 5 V abweichen darf. Da aber die Netzspannung schon um 10% schwanken kann, sind besondere Stabilisierungs-Schaltungen nötig, um diese Forderung einzuhalten. Eine einfache Methode hierzu zeigt *Abb. 1.5.4.1:* Mit einem Widerstand und einer Z-Diode wird zunächst eine Hilfsspannung von z.B. 5,7 V erzeugt. Ein entsprechend belastbarer Leistungstransistor stellt an seinem Emitter dann 5 V zur Verfügung, da an seiner Basis-Emitter-Strecke rund 0,7 V abfallen. Die unstabilisierte Eingangsspannung muß mindestens etwa 2 V höher als die gewünschte Ausgangsspannung sein, da die Regelung sonst nicht korrekt arbeitet.

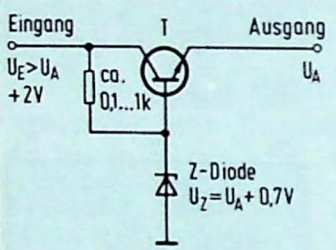


Abb. 1.5.4.1 Einfache Spannungs-Stabilisation mit Z-Diode und Leistungstransistor als Längsregler

Integrierte Spannungsregler

Eine solche Einfachst-Schaltung hat zwei Nachteile: Zum einen ist sie nicht kurzschlußsicher; der Transistor würde beschädigt, wenn ein zu hoher Strom entnommen wird. Zum anderen ist die erreichbare Stabilität wegen der geringen Regelverstärkung des einzigen Transistors nicht allzu hoch.

Es gibt deshalb eine Reihe integrierter Spannungsregler auf dem Markt. Sie enthalten im wesentlichen:

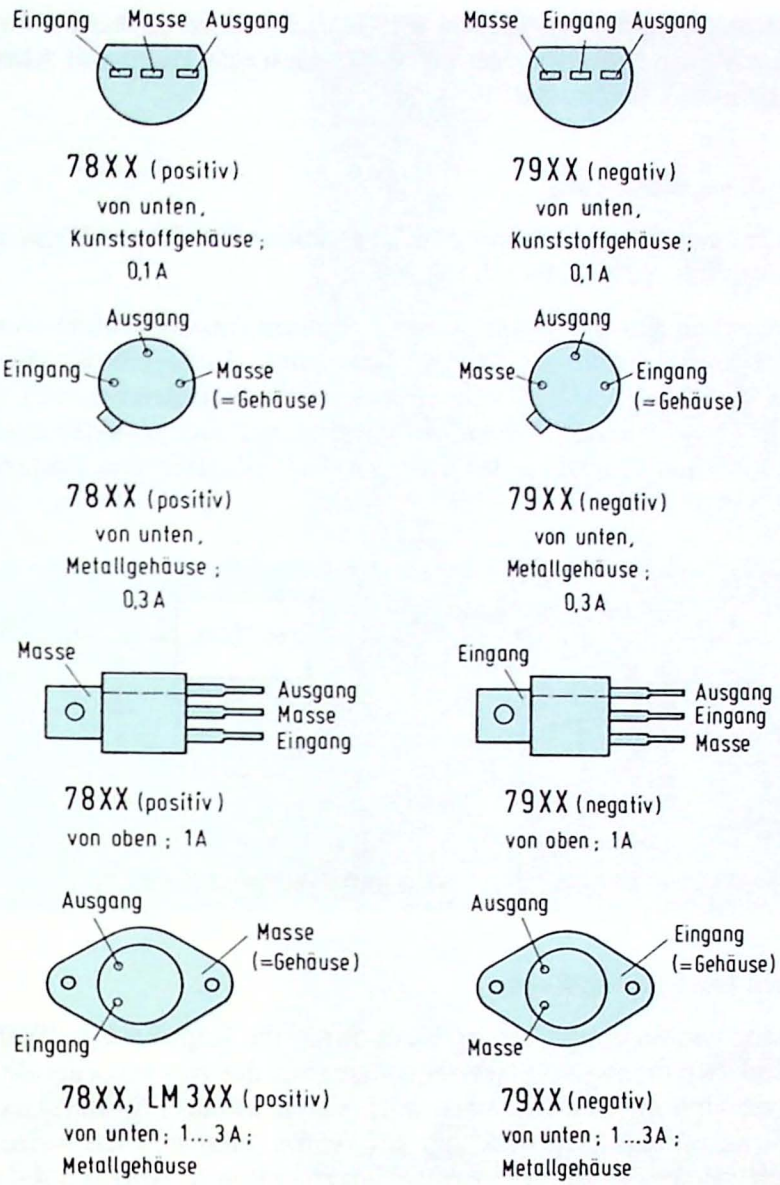


Abb. 1.5.4.2 Gehäuseformen integrierter Spannungsregler

1. eine hochstabile Referenzspannungsquelle,
2. einen integrierten Regelverstärker,
3. einen Leistungstransistor entsprechender Belastbarkeit,
4. eine Schutzschaltung zur Begrenzung des Ausgangsstroms abhängig von der Chip-Temperatur.

Die Gehäuseformen solcher Spannungsregler zeigt die *Abb. 1.5.4.2.* Die Anschlußbelegung ist unterschiedlich, je nachdem, ob eine positive oder negative

Ausgangsspannung stabilisiert werden soll. Insbesondere ist darauf zu achten, daß bei den Reglern für negative Spannungen das Gehäuse nicht mit Masse (Bezugspotential) verbunden werden darf.

Netzteil-Schaltungsauslegung

Eine praktisch ausgeführte Schaltung für ein kleines 5-V-Netzteil mit einer Belastbarkeit von etwa 0,8 A zeigt *Abb. 1.5.4.3.*

Die Belastbarkeit des Trafos sollte nie zu knapp dimensioniert werden, da der Strom in ihm nur impulsweise zum Aufladen des „Lade-Elkos" (hier 4700 µF) fließt. Dieser Kondensator lädt sich über den Brückengleichrichter (B40C1000: 40 V/1000 mA) auf etwa den 1,4fachen Effektivwert der sinusförmigen Wechselspannung an der Sekundärseite des Trafos auf (Verhältnis von Spitzen- zu Effektivwert einer Sinusspannung).

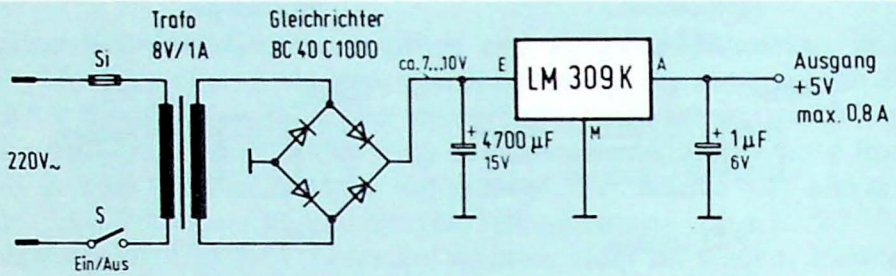


Abb. 1.5.4.3 Schaltung eines einfachen Netzteils mit 5 V Ausgangsspannung

Unterdrücken von Störimpulsen

Die Erzeugung von Störimpulsen im Netz durch die impulsweise Trafo-Belastung und auch die Übertragung von Netzstörungen auf die zu versorgende Logikschaltung kann vermindert werden, wenn man einen kleinen Widerstand (ca. 0,5... 1 Ohm) in Serie zur Sekundärwicklung des Trafos schaltet. Eine weitere Verbesserung läßt sich mit speziellen Netz-Entstörfiltern erzielen. *Abb. 1.5.4.4* zeigt ein solches Filter: Es enthält eine Spule (deren Innenwiderstand bekanntlich mit steigender Frequenz zunimmt) und Kondensatoren, die für hochfrequente Impulsstörungen praktisch einen Kurzschluß darstellen. Die Innenschaltung eines solchen Filters ist in *Abb. 1.5.4.5* nochmals dargestellt.

Schaltnetzteile

Versucht man, nach dem beschriebenen Verfahren Netzteile für Ströme von mehr als 3...5 A aufzubauen, so sind dazu ein großer und schwerer Netztransformator sowie umfangreiche Kühleinrichtungen notwendig: Die in Wärme umgesetzte Verlustleistung errechnet sich aus dem Produkt von Ausgangsstrom und der am Stabi-

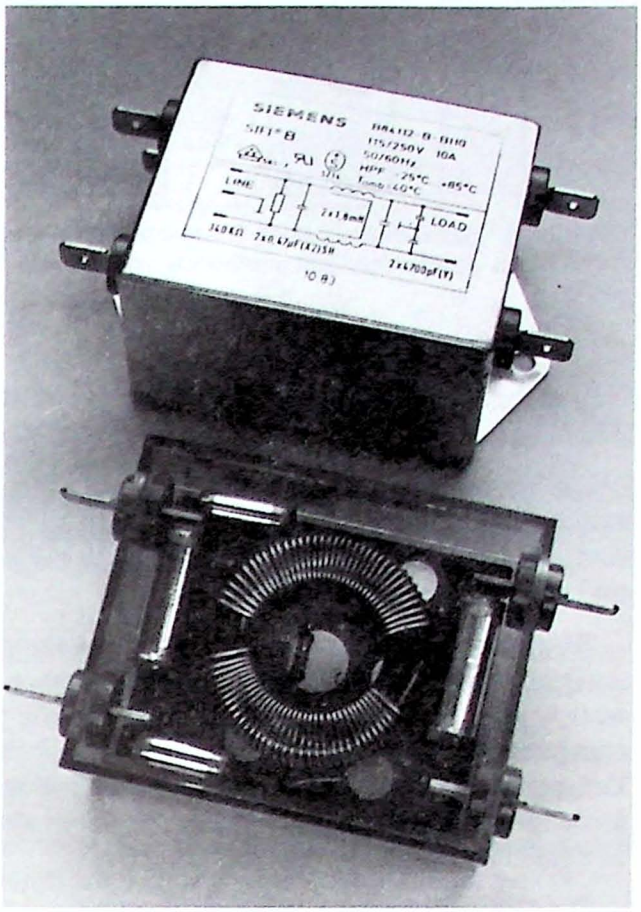


Abb. 1.5.4.4
Netzfilter zum Fernhalten von Störimpulsen aus dem 220-V-Netz

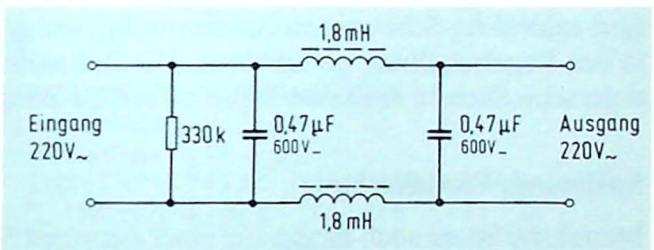


Abb. 1.5.4.5
Innenschaltung eines Netzfilters. Die beiden Induktivitäten sind bifilar auf einen gemeinsamen Ringkern gewickelt

lisator-IC (bzw. Regeltransistor) abfallenden Differenzspannung. Bei einem typischen Netzteil für 5 V/5 A liegen am Stabilisator-Eingang meist rund 8 V; die Verlustleistung beträgt also:

$$P = (8\ V - 5\ V) * 5\ A = 15\ W$$

Diese Leistung kann nur mit großen Kühlblechen oder -körpern bzw. mit einem Ventilator abgeführt werden.

Eine Alternative zeigt *Abb. 1.5.4.6* als Blockschaltbild. Ein Gleichrichter macht aus der 220-V-Netzspannung zunächst eine Gleichspannung von rund 300 V. Sie wird von einem Oszillator, der mit einigen kHz eine gegenüber der 50-Hz-

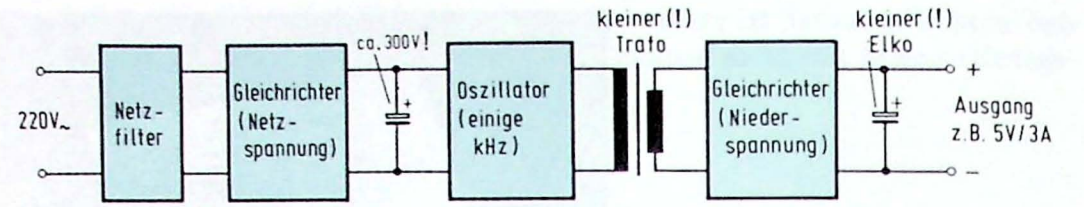


Abb. 1.5.4.6 Blockschaltbild eines Schaltnetzteils. Im Vergleich zu herkömmlichen, längsgeregelten Netzteilen ist ein kleinerer Trafo erforderlich, und auch die als Wärme abgegebene Verlustleistung ist geringer

Netzspannung recht hohe Frequenz erzeugt und dessen Bauelemente natürlich die 300 V aushalten müssen, wieder in eine Wechselspannung umgewandelt.

Da der nötige Trafo bei hohen Frequenzen recht klein bleiben kann, erreicht man dadurch eine erhebliche Gewichtseinsparung. Ebenso ist der sekundärseitig nötige Elko kleiner als bei einem herkömmlichen Netzteil, weil er ja einige kHz und nicht 50 Hz aussieben muß.

Wenn man nun noch den Oszillator mit einer Regelschaltung stoppt, sobald die Ausgangsspannung den Sollwert überschreitet, und neu startet, sobald der Sollwert wieder unterschritten wird, läßt sich die Regelung nahezu ohne Verlustleistung realisieren. Natürlich ist es nicht möglich, einen Oszillator mit einem Wirkungsgrad von 100% zu bauen, und er selbst wird daher schon ein wenig Verlustleistung und damit Wärme produzieren — aber nur etwa 10...20% eines herkömmlichen Netzteils.

Die Dimensionierung von Schaltnetzteilen ist allerdings nicht unkritisch, und es sind zahlreiche Schutzmechanismen nötig, um größere Schäden bei einem Defekt in der Regelschaltung zu verhüten. Die Industrie bietet jedoch Schaltnetzteile für unterschiedlichste Anforderungen als fertige Baugruppen an.

Spannungs-Vervielfachung

Manchmal ist es auch nötig, aus einer geringen Spannung (z.B. 5 V) eine wesentlich höhere zu erzeugen (z.B. 25 V zum Programmieren eines EPROMs). Das ist auch bei batteriebetriebenen Geräten oder dann, wenn keine zusätzliche Trafowicklung im Netzteil zur Verfügung steht, durchaus kein großes Problem.

Abb. 1.5.4.7 enthält eine solche Schaltung unter Verwendung eines speziell dafür entwickelten ICs. Es enthält einen steuerbaren Oszillator, eine Regelschaltung, eine Strombegrenzungs-Schutzschaltung und eine Gleichrichterdiode.

Der Oszillator läßt durch die an den Pins 13 und 10 angeschlossene Spule impulsweise Strom fließen. Diese Impulse bauen in der Spule jedesmal ein Magnetfeld auf, das sich in den Impulspausen schlagartig abbaut und dabei eine hohe „Rückschlagspannung" erzeugt. Sie lädt über die Diode den Elko am Ausgang auf eine hohe Spannung auf. Überschreitet diese Ausgangsspannung den Sollwert, so

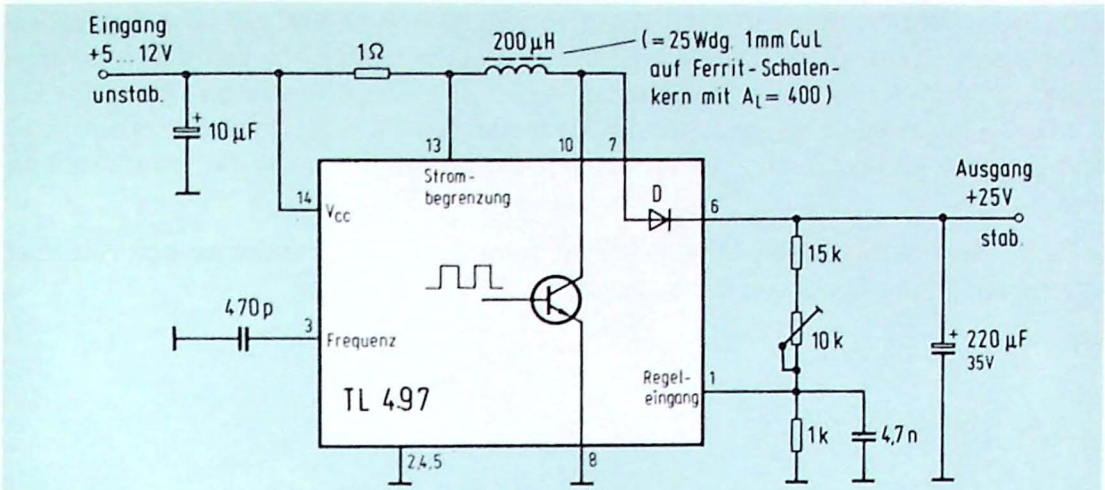


Abb. 1.5.4.7 Mit der integrierten Schaltung TL 497 von Texas Instruments kann man z.B. die zum Programmieren von EPROMs nötige Spannung von 25 V aus nur 5 V gewinnen

stoppt die Regelschaltung den Oszillator vorübergehend. Der Ausgang ist mit einigen mA belastbar, was z.B. zum Programmieren eines EPROMs durchaus ausreicht.

Netzteil-Dimensionierung in Basic

Zurück zu dem herkömmlich aufgebauten Netzteil mit „Längsregler", wie man den längs am Ausgang liegenden Spannungsregler auch nennt. Mit dem kurzen Basic-Programm in *Abb. 1.5.4.8* kann man überschlagsmäßig alle wichtigen Daten zur

```
10REM NETZTEIL-BERECHNUNG F.INTEGR.SPANNUNGSREGLER
20INPUT"Gewünschte Ausgangsspannung/V:";UA
30INPUT"Maximaler Ausgangsstrom/A:";IA
40PRINT:PRINT"Ladeelko: ";INT(IA*4000);" µF/";INT(UA+6);" V"
50PRINT"Gleichrichter min.";INT(UA*1.6);" V/";IA*1.2;" A"
60PRINT"Trafo: ";INT((UA+5)/1.3);" V/";INT(IA*12+1)/10;" A"
70PRINT"Primärsicherung: ";INT(IA*UA/10)/10+.1;" A"
80PRINT"Verlustleistung: ";INT(1+(.3*UA+3)*IA);" W"
90END


>RUN
Gewünschte Ausgangsspannung/V:5
Maximaler Ausgangsstrom/A:0.8

Ladeelko: 3200 µF/11 V
Gleichrichter min.7 V/.96 A
Trafo: 7 V/1 A
Primärsicherung: .1 A
Verlustleistung: 4 W
```

Abb. 1.5.4.8
Basic-Programm zur überschlägigen Netzteil-Dimensionierung, auf einem Apple-II geschrieben. Nach RUN ist eine Beispiel-Berechnung angegeben

Dimensionierung ermitteln. Gibt das Programm beispielsweise eine Ladeelko-Größe von 3200 µF aus, so wird man den nächstgrößeren erhältlichen Wert 3300 µF wählen (oder sicherheitshalber 4700 µF). Die ermittelten Daten für die Ladeelko-Spannungsfestigkeit und die Verlustleistung des Reglers gelten natürlich nur dann, wenn der Trafo auch tatsächlich die ermittelte Sekundärspannung aufweist.

Das Programm berücksichtigt mögliche Toleranzen von Bauelementen oder der Netzspannung und geht von der Schaltung in Abb. 1.5.4.3 aus.

1.5.5 Sensoren

Sensoren dienen dazu, um aus nichtelektrischen Größen elektrische Signale zu machen, die sich z.B. mit einem Mikrocomputer auswerten lassen. Typische Sensoren sind:

Meßgröße	Geeigneter Sensor
Helligkeit	Fotodiode, Fototransistor, Fotowiderstand (LDR = Light Dependant Resistor)
Temperatur	Heißleiter (NTC = Negative Temperature Coefficient), Kaltleiter (PTC = Positive Temperature Coefficient), Bimetall-Schaltkontakt
Schall	Dynamisches, Piezokristall- und Elektret-Mikrofon
Wegstrecke	Dehnungsmeßstreifen (bis 1 mm), optische Abtastung, Ultraschall-Laufzeitmessung
Drehzahl	Tachogenerator, optische Abtastung, Nockenkontakt
Magnetfeld	Hall-Generator, magnetfeldabhängiger Widerstand
Luftfeuchte	Kapazitiver Feuchtigkeits-Sensor
Gasdruck	Silizium-Brückensensor

Je nachdem, ob nur Über- oder Unterschreitung eines Grenzwertes geprüft oder der tatsächliche Wert gemessen werden soll, erfolgt die Verbindung des Sensors mit dem Mikrocomputer entweder über einen Schmitt-Trigger (siehe „TTL-Schaltungen") oder über einen Analog-Digital-Umsetzer. Nun aber einige Beispiele, wie man unterschiedliche nichtelektrische Größen auswerten kann.

Strichcode-Leser

Der Strichcode (Barcode) wird in einem späteren Kapitel noch näher besprochen. Er findet sich z.B. auf Lebensmittelpackungen. Die Produktnummer wird in Form dicker und dünner Striche markiert. Die am häufigsten anzutreffende Vorrichtung zum Lesen von Strichcode ist der Lesestift *(Abb. 1.5.5.1)*. In seinem Gehäuse finden sich eine Leuchtdiode und eine Fotodiode, die hinter einer Optik so angeord-

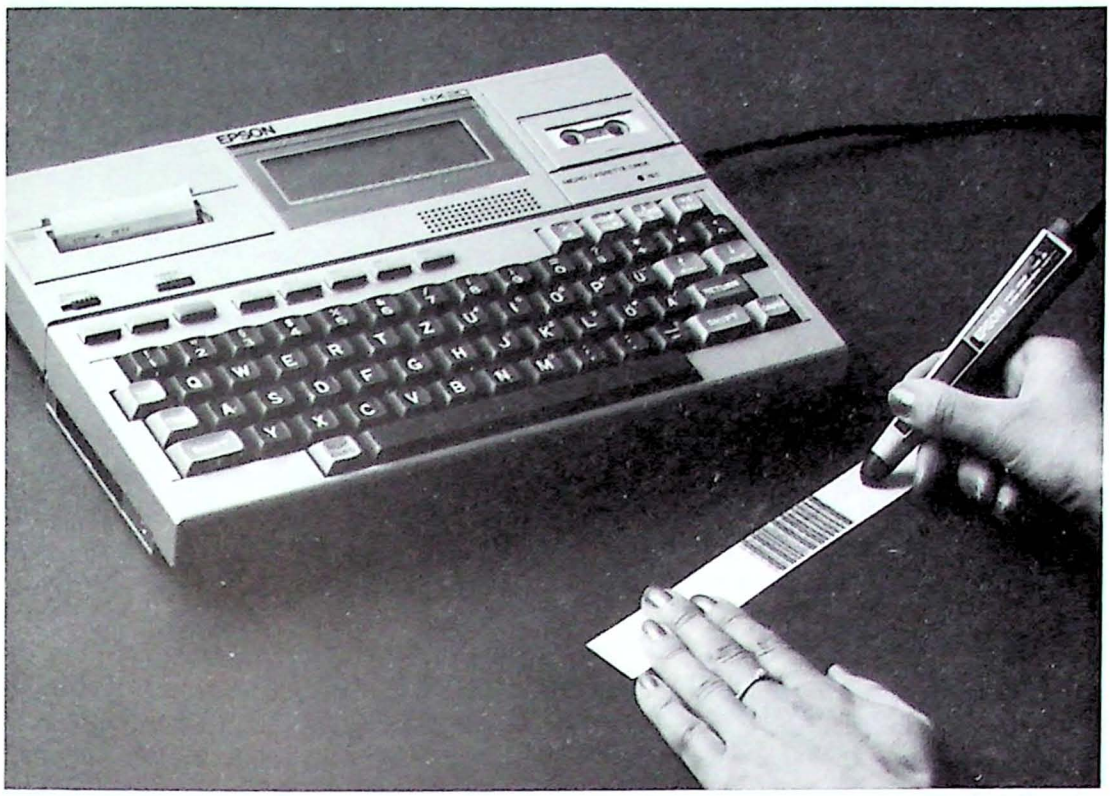


Abb. 1.5.5.1 Strichcodeleser, hier an einen tragbaren Computer HX-20 angeschlossen

net sind, daß beide auf den gleichen Brennpunkt fokussiert sind. Sie stellen zusammen also eine Reflex-Lichtschranke dar.

Von unbedrucktem, weißem Papier wird das Licht der LED zur Fotodiode reflektiert, und diese stellt für elektrischen Strom einen geringen Widerstand dar. Über schwarz bedrucktem Papier fällt dagegen kaum Licht auf sie, und ihr Innenwiderstand wird sehr hoch. Daraus läßt sich leicht eine Spannungsänderung ableiten, und wenn der Lesestift mit einigermaßen konstanter Geschwindigkeit über den Strichcode bewegt wird, entspricht die Breite eines schwarzen Balkens unmittelbar der Zeit, in der die Fotodiode nicht leitet — was sich leicht durch Zeitschleifen von einem Programm im angeschlossenen Mikrocomputer auswerten läßt.

Gasdruck-Messung

Die Messung von Luft- oder Gasdruck ist mit Hilfe eines Silizium-Brückensensors möglich (z.B. KPY 10 von Siemens, *Abb. 1.5.5.2).* Er besteht intern aus vier Widerständen von je etwa 7 kΩ. Sie sind so geschaltet, daß sie normalerweise (bei keinem äußeren Druck) im „Gleichgewicht" sind: Die Ausgangsspannung ist Null. Wirkt Druck auf die Widerstände ein, so verbiegen sie sich und ändern dabei ihren Wert. Beispielsweise werden die beiden Widerstände rechts oben und links unten klei-

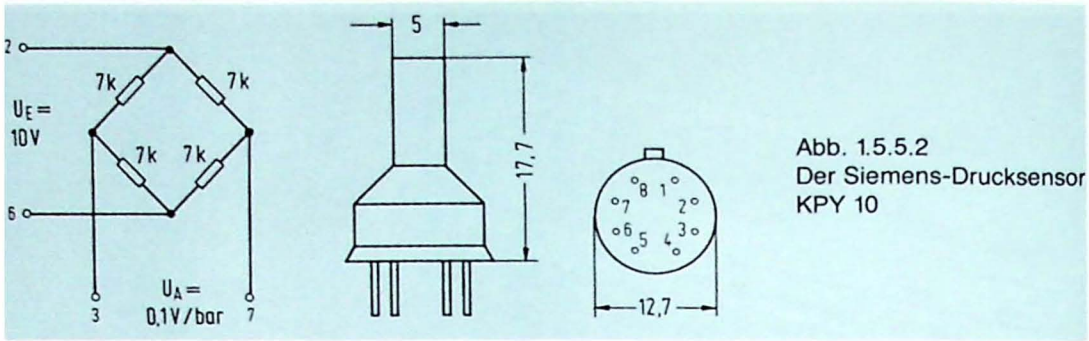


Abb. 1.5.5.2
Der Siemens-Drucksensor KPY 10

ner, die anderen beiden größer, und am Ausgang entsteht eine Spannung, die dem Druck direkt proportional ist (rund 0,1 V pro bar bei 10 V Eingangsspannung). Der Anschluß ist auch über einen Luftschlauch leicht möglich, weil das Gehäuse einen Rohrstutzen besitzt. Der Druck darf allerdings etwa 2 bar nicht überschreiten, da sonst bleibende Verformungen auftreten können.

Temperaturmessung

Im Prinzip ist mit jeder Halbleiterdiode schon eine Temperaturmessung möglich. Denn die Schwellspannung nimmt z.B. bei einer Siliziumdiode mit rund 2,2 mV pro Grad Celsius ab. Man braucht also nur einen konstanten Strom durch die Diode zu leiten und die Spannung zu messen. Leider treten aber erhebliche herstellungsbedingte Toleranzen der Schwellspannung auf, und die Spannungsänderung ist vergleichsweise gering. Das stört insbesondere deshalb, weil die nötige Verstärkerschaltung, die ja selbst mit Transistoren aufgebaut ist, ebenfalls einen Temperaturkoeffizienten aufweist, der erst umständlich kompensiert werden muß, damit er das Meßergebnis nicht verfälscht.

Es gibt daher spezielle Temperatursensoren, die in dem jeweils interessierenden Bereich (z.B. 40...100 Grad Celsius) eine recht hohe Änderung ihres Widerstandswertes pro Grad besitzen und dadurch die Meßschaltung unkritischer machen, z.B. der Siemens-Silizium-Kaltleiter KTY 10, der speziell für Raumtemperatur-Messungen geeignet ist und bei 25 Grad Celsius einen Widerstand von etwa 2 kΩ aufweist *(Abb. 1.5.5.3)*. Dieser Wert nimmt bei steigender Temperatur stark zu.

In jedem Fall sollte man bei Temperaturmessungen darauf achten, daß der durch den Fühler fließende Meßstrom nicht selbst eine verfälschende Erwärmung

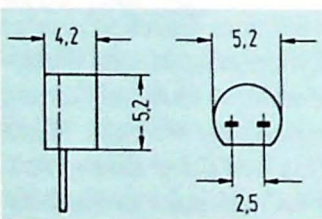


Abb. 1.5.5.3
Silizium-Temperaturfühler (KTY 10 von Siemens)

des Bauelements hervorruft. Beim KTY 10 sollte man daher nur etwa 2 V anlegen, so daß ein Strom von etwa 1 mA fließt; das entspricht einer Verlustleistung von 2 mW.

Bei Heißleitern ist das Verhalten des Widerstandswertes genau umgekehrt: Er nimmt bei steigender Temperatur ab. Solche Bauelemente eignen sich ebenfalls recht gut zur Temperaturmessung, allerdings ist ihre Widerstandsänderung gewöhnlich nicht so steil wie bei Kaltleitern.

Dehnungsmeßstreifen

Um Verformungen an Materialien zu kontrollieren, gibt es sogenannte Dehnungsmeßstreifen. Bei ihnen ist eine Widerstandsschicht auf ein flexibles Trägermaterial aufgebracht *(Abb. 1.5.5.4)*. Zur Messung werden sie einfach auf das zu untersuchende Werkstück aufgeklebt. Dehnt sich das Werkstück nun aus, so wird die Widerstandsbahn ein wenig länger, und der Widerstandswert nimmt um einen kleinen Betrag zu. So kann man noch Verformungen nachweisen, die nur den Bruchteil eines Millimeters ausmachen.

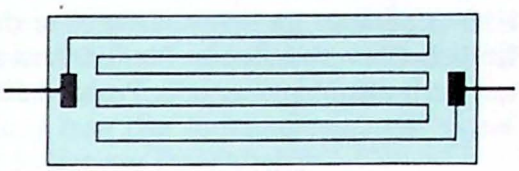


Abb. 1.5.5.4 Aufbau eines Dehnungsmeßstreifens. Auf dem dehnbaren Trägermaterial befindet sich eine schlangenförmige Widerstandsbahn

Luftfeuchtigkeits-Sensor

Für die Messung der relativen Luftfeuchtigkeit hat Valvo einen kapazitiven Sensor entwickelt *(Abb. 1.5.5.5)*. Er besteht aus einer beidseitig metallisch bedampften Folie, die unter dem Einfluß der Luftfeuchtigkeit ihr dielektrisches Verhalten ändert. Die Kapazität des Bauelements beträgt etwa 120 pF und nimmt pro Prozent Feuchtigkeit um etwa 0,4 pF zu. Die Auswertung kann z. B. mit einem Oszillator erfolgen, der den Feuchtesensor als frequenzbestimmenden Kondensator verwendet und mit der relativen Feuchte die erzeugte Frequenz ändert. Eine typische Anwendung ist die automatische Regelung der optimalen Luftfeuchte in Gewächshäusern.

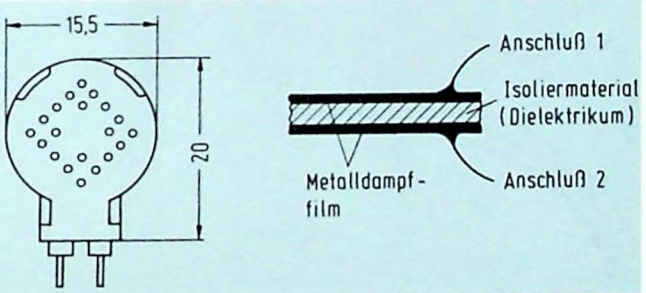


Abb. 1.5.5.5
Aufbau und Gehäuseform eines kapazitiv arbeitenden Feuchtigkeitssensors von Valvo

Magnetfeld-Messung

Für die Messung von Magnetfeldern gibt es zwei Arten von Halbleiter-Sensoren. Ein magnetfeldabhängiger Widerstand (z.B. FP30-L100-E) eignet sich nur für recht starke Felder. Dabei macht man sich den Effekt zunutze, daß ein Magnetfeld sozusagen die Elektronen aus ihrer Richtung bringt — sie machen einen Umweg und der Widerstand (etwa 100 Ω) erhöht sich bis etwa um den Faktor 7. Allerdings muß dabei der Temperaturkoeffizient von rund −0,54 Prozent pro Grad Celsius kompensiert werden.

Um die Empfindlichkeit zu erhöhen, gibt es auch vier auf einem Chip integrierte magnetfeldabhängige Widerstände. Sie sind ähnlich wie bei dem schon besprochenen Drucksensor als Brückenschaltung angeordnet. Diese gerät in einem Magnetfeld aus dem Gleichgewicht, und es entsteht eine dem Feld direkt proportionale Ausgangsspannung.

Eine interessante Ausführung eines solchen Hall-Generators zeigt *Abb. 1.5.5.6.* Sie dient als kontaktloser Unterbrecher in der Zündelektronik eines Kraftfahrzeugs. Auf der einen Seite befindet sich ein Permanentmagnet, auf der anderen ein Hall-Generator. Im Spalt dazwischen dreht sich die Nockenscheibe, die mit der Kurbelwelle verbunden ist. Natürlich ist das Bauelement auch z.B. als Endabschalter, Drehzahlaufnehmer oder Positionsfühler anwendbar.

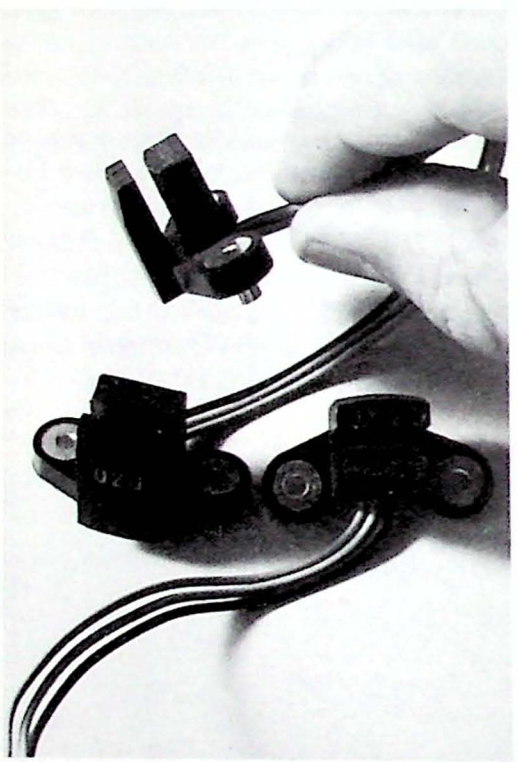


Abb. 1.5.5.6
Hall-Generator-Gabel für die Zündung in Kraftfahrzeugen (Siemens)

1.6 Aufbautechniken

1.6.1 Fädel- und Wire-Wrap-Verdrahtung

Die einzelnen Bauelemente eines Mikrocomputers wurden nun ausführlich besprochen. Bliebe noch zu klären, wie sie eigentlich miteinander verbunden werden.

Beim Bau von Einzelstücken und, während der Vorversuche zu einer Serienfertigung, von Prototypen haben sich im wesentlichen zwei Aufbautypen durchgesetzt: Fädeln und Wire-Wrap. Die Anfertigung geätzter Platinen (siehe nächstes Kapitel) würde sich hier nämlich (noch) nicht lohnen.

Zum Fädeln verwendet man dünnen Kupferlackdraht. Dabei handelt es sich um eine Drahtsorte mit 0,1...0,3 mm Durchmesser, deren Kupferleiter von einer sehr dünnen Kupferlackschicht isoliert ist. Bei Temperaturen von mehr als 200 Grad Celsius, wie sie beim Löten auftreten, löst sich diese Lackschicht von selbst auf, so daß ein mühsames Abisolieren der Enden nicht erforderlich ist.

Die einzelnen Bauelemente werden zunächst auf eine sogenannte Lochrasterplatte gesteckt *(Abb. 1.6.1.1)*. Besitzt diese keine Lötaugen, so werden die auf der Lötseite herausstehenden Anschlußdrähte ein wenig umgebogen, so daß die Teile nicht mehr herausfallen können, wenn man die Platte zum Löten umdreht. Bei Lochrasterplatten mit Lötaugen genügt ein kleiner Lötzinn-Tupfer, um das Herausfallen zu vermeiden. Anschließend kürzt man die Anschlußbeine mit einem Seitenschneider so, daß sie noch etwa zwei Millimeter herausstehen.

Dann kann man sich an die Verdrahtung mit Kupferlackdraht machen. In der Computertechnik reicht hierzu ein Lötkolben mit etwa 15...30 W Leistung bzw. ein Typ mit automatischer Temperatur-Regelung (z.B. Weller-Magnastat). Insbesondere ist eine kleine Lötspitze erforderlich. Das verwendete Lötzinn sollte als Flußmittel lediglich Kollophonium, aber keinesfalls eine ätzende Säure enthalten. Diese könnte nämlich die Isolationseigenschaften der Lochrasterplatte (oder geätzten Platine) langfristig beeinträchtigen. Das richtige Löten ist eine Wissenschaft für sich; kurz zusammengefaßt sieht das Verlöten einer einzigen Verbindung so aus

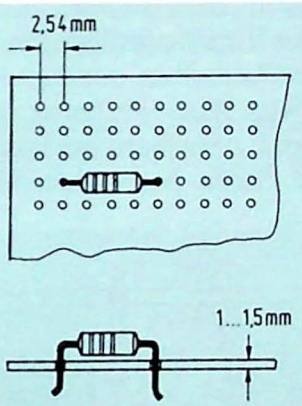


Abb. 1.6.1.1
Versuchsschaltungen baut man zweckmäßig auf sogenannten Lochrasterplatten auf

(Gesamtdauer je nach Übung etwa ein bis zwei Sekunden, keinesfalls länger, sonst schmort die Platine!):

1. Nur mit wirklich heißem Lötkolben arbeiten! Nur so ist ein schnelles, sauberes Löten möglich.
2. Lötstelle mit dem Lötkolben kurz vorwärmen, damit das Lot nachher gut fließen kann.
3. Anzulötenden Draht und (!) frisches Zinn von der Rolle an die Lötstelle führen und verlöten.
4. Draht erst dann loslassen, wenn das Zinn erkaltet ist, um mechanische Spannungen in der Lötstelle zu vermeiden.

Die einzelnen Lackdraht-Verbindungen können entweder kreuz und quer über die Platine oder gruppenweise in Bündeln geführt werden. Bei hohen Taktfrequenzen ist allerdings eine Kreuz-und-Quer-Verdrahtung sinnvoller als die auf den ersten Blick vielleicht schöner aussehende Bündel-Verdrahtung, da sie mit kürzeren Leitungslängen auskommt und zu geringeren Verdrahtungskapazitäten führt.

Besonderes Augenmerk verlangen die Masse- und Versorgungsspannungsleitungen. Da auf ihnen erhebliche Ströme fließen, ist es zweckmäßig, sie aus dickerem Schaltdraht auszuführen. Für diesen Zweck gibt es auch fertige aus Blech gestanzte Stege, die auf der Bauelementeseite der Platine für die nötigen Verbindungen sorgen. Darüber hinaus ist zu bedenken, daß TTL- und CMOS-ICs der Spannungsversorgung immer während des Umschaltens einen impulsförmigen Strom entnehmen. Dieser Impuls kann sich störend auf andere ICs auswirken. Deshalb ist es ratsam, auf der Platine in regelmäßigem Abstand zwischen Masse und Versorgungsspannung Abblock-Kondensatoren unterzubringen (ca. 0,1...1 µF), die diese Impulse kurzschließen. Die Hersteller dynamischer RAM-Bausteine empfehlen solche Kondensatoren sogar parallel zu jedem dieser ICs.

Es ist während des Fädelaufbaus unbedingt ratsam, die genaue Anschlußzuordnung auch bei jenen ICs ins Schaltbild einzutragen, die mehrere gleichartige Logikgatter in einem Gehäuse besitzen. Das vereinfacht eine spätere Fehlersuche ganz erheblich.

Außer dem „Fädeln" ist manchmal auch die Wire-Wrap-Verdrahtung üblich, insbesondere bei Einzelanfertigungen und Kleinserien. Dabei wird nicht mit Kupferlackdraht und Lötkolben, sondern mit kunststoffisoliertem Schaltdraht und einem speziellen Wire-Wrap-Werkzeug gearbeitet. Dieses Werkzeug beinhaltet eine Drahtrolle, eine Abisoliervorrichtung und einen Drehstift, der den Schaltdraht um den aus der Platine herausstehenden Anschlußdraht eines Bauelements so fest

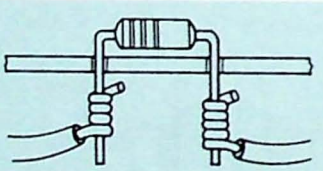


Abb. 1.6.1.2
Bei der Wire-Wrap-Technik werden die Drahtenden nicht verlötet, sondern nur mit Hilfe einer speziellen Wrap-Zange fest verdrillt

herumwickelt, daß ein sicherer Kontakt gewährleistet ist *(Abb. 1.6.1.2)*. Die Bauelement-Anschlüsse müssen dabei allerdings mindestens etwa 8 mm lang bleiben und auch die nötige Festigkeit aufweisen. Herkömmliche Widerstände, Kondensatoren und auch ICs ohne Fassungen halten diese Bedingungen nicht ein, so daß die Wire-Wrap-Technik heute vorzugsweise bei der Verbindung von ganzen Platinen untereinander verwendet wird. Die Platinen erhalten auf einer Seite eine dafür geeignete Steckleiste, die in einer Trägerleiste (z.B. Europakarten-Träger) befestigt und per Wire-Wrap verdrahtet wird.

1.6.2 Gedruckte Schaltungen

Für eine Serienfertigung wäre es viel zu umständlich, jedes Gerät einzeln mit Schalt- oder Lackdraht zu verdrahten. Deshalb verwendet man dafür sogenannte gedruckte Platinen. Eigentlich müßte es „geätzte Platinen" heißen; gedruckt ist nämlich lediglich die Vorlage für die Leiterbahnführung.

Eine solche Platine entsteht, nachdem in einem Lochraster-Probeaufbau die Funktionsfähigkeit der Schaltung unter Beweis gestellt wurde, folgendermaßen:

1. Die nötigen Leiterbahnen werden mit Hilfe eines Rechners mit angeschlossenem Plotter oder mit Tusche bzw. selbstklebenden schwarzen Bahnen im Maßstab 2:1 oder 4:1 sauber gezeichnet.
2. Von dieser Vorlage wird durch fotografische Verkleinerung ein transparenter Film hergestellt.
3. Eine Platte aus Isoliermaterial (meist glasfaserverstärktes Epoxid-Kunstharz), die mit einer dünnen Kupferfolie beklebt ist, wird mit einer fotoempfindlichen Schicht besprüht.
4. Der transparente Film wird auf die beschichtete Platine gelegt und einige Sekunden bis Minuten (je nach Fotoschicht) unter einer UV-Lampe belichtet.
5. Die belichtete Platine wird in einem Bad aus schwacher Natronlauge entwikkelt. Dabei löst sich die Fotoschicht an allen belichteten Stellen, also dort, wo keine Leiterbahnen sein sollen.
6. Ein Ätzbad (z.B. Wasser mit Eisen-III-Sulfat) löst die Kupferschicht jetzt überall dort herunter, wo sie nicht durch stehengebliebenen Fotolack geschützt ist. Die Leiterbahnen bleiben also stehen.
7. Nun wird noch der restliche Fotolack heruntergewaschen (dieser Schritt entfällt bei lötfähigen Fotolack-Sorten).
8. Die Löcher für die Bauelemente-Anschlüsse werden gebohrt.
9. Eventuell werden die Kupfer-Leiterbahnen noch verzinnt. Dann kann die Platine mit Bauelementen bestückt und gelötet werden.

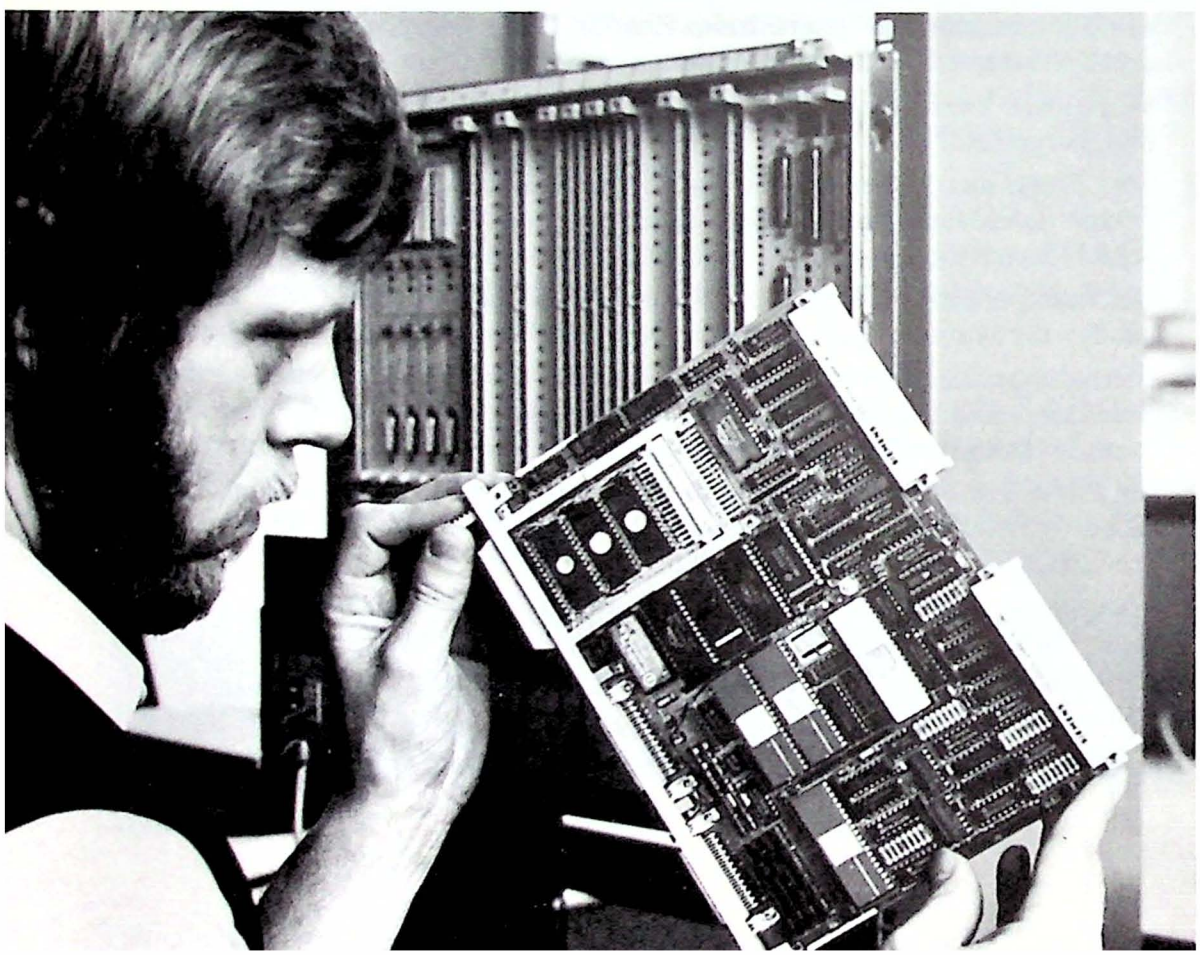
Alternativ zu der Beschichtung der Platine mit Fotolack und dessen Belichtung ist es auch möglich, die Kupferauflage z.B. im Offset-Verfahren mit ätzresistenter Farbe zu bedrucken, was insbesondere bei der Fertigung größerer Serien rationeller ist.

Die Vielzahl der hier genannten Arbeitsgänge hört sich aufwendig an, lohnt sich aber schon bei Kleinserien von nur wenigen Stück. Denn die Handverdrahtung ist fehleranfällig und unübersichtlich.

Für das Anfertigen des Layouts, also der Vorlage (Schritt 1), gibt es bereits spezielle Rechenprogramme, denen man die Schaltung codiert eingeben kann und die dann mehr oder weniger automatisch sich um eine kreuzungsfreie Leiterbahnführung kümmern.

Bei komplexen Schaltungen muß man beidseitig beschichtete Platinen verwenden und die Leiterbahnen von Oberseite (Bestückungsseite) und Unterseite (Lötseite) miteinander an den Bauelemente-Löchern verbinden. Dies geschieht, indem nach dem Bohren die Löcher auf galvanischem Wege innen metallisiert werden. Da der Aufwand hierzu nicht unerheblich ist, kommt dieses Durchkontaktieren nur industriell in Frage.

Abb. 1.6.2.1 Doppel-Europakarte



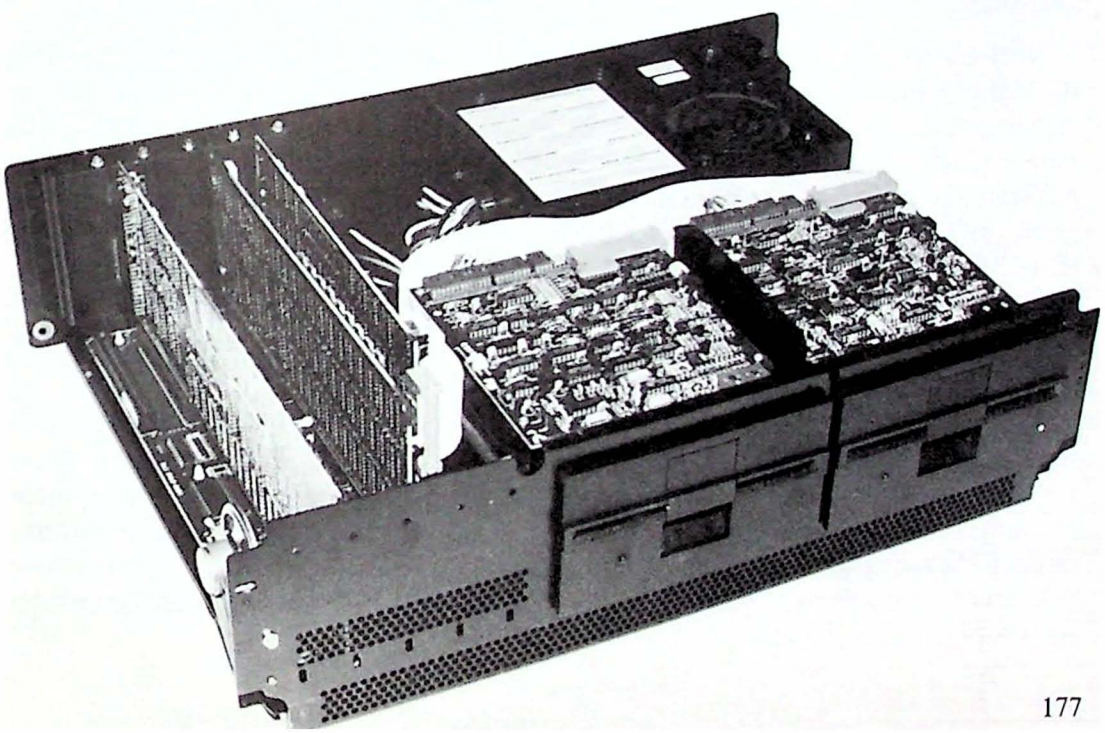
Eine Notlösung für die Inbetriebnahme doppelseitig geätzter Platinen ohne galvanische Durchkontaktierung ist das Verwenden von kleinen Hohlnieten in den (entsprechend größer aufgebohrten) Löchern oder das Verlöten aller Bauelemente auf der Ober- und Unterseite. Bei Bauelementen, die das nicht zulassen (z.B. dicht auf der Platine sitzende IC-Fassungen), gibt es noch die Möglichkeit, durch jedes Loch zunächst einen dünnen Draht zu stecken, ihn oben und unten zu verlöten und erst dann das Bauelement einzusetzen.

Für Standard-Platinengrößen gibt es auch bereits vorgefertigte Gehäuse und Baugruppenträger, so etwa für die Europakarte ($100 \times 160\,mm^2$) und die Doppel-Europakarte *(Abb. 1.6.2.1)*. Es gibt auch die Möglichkeit, kleinere Platinen auf eine gemeinsame Mutterplatine zu stecken, wie das z.B. bei den „Slots" (Platinen-Steckleisten) im Apple-II oder IBM-PC *(Abb. 1.6.2.2)* realisiert ist.

Oftmals möchte der Platinenhersteller dem Kunden die Möglichkeit zu einer individuellen Schaltungs-Erweiterung offenlassen. Dann kann ein Teil der Platine als nachträglich frei verdrahtbare Lochraster-Fläche ausgeführt werden *(Abb. 1.6.2.3)*.

Für das Bestücken und Verlöten von Bauelementen in geätzten Platinen gilt prinzipiell das schon im letzten Kapitel Gesagte. Um Kurzschlüsse durch versehentliche Lötzinn-Spritzer zu vermeiden, ist auf manche Platinen auch ein spezieller Lötstopplack aufgebracht, der nur die zu verlötenden „Augen" freiläßt. Auf jeden Fall sollte man sich hüten, mit dem Lötkolben allzu lange auf einer Leiterbahn „herumzubraten", da sie sich dabei leicht vom Trägermaterial lösen kann.

Abb. 1.6.2.2 Das Innere des IBM-PC. Auf der linken Seite sind Steckkarten in drei „Slots" zu sehen



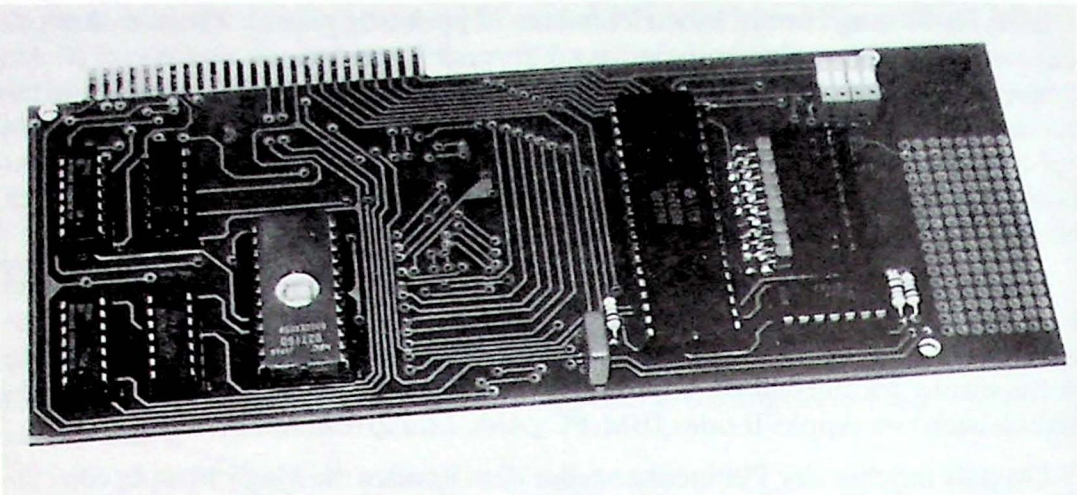


Abb. 1.6.2.3 Steckkarte für den Apple-II, hier mit einem EPROM 2716 und einem RIOT-Baustein 6532

Bei größeren Serien werden die Platinen allerdings nicht von Hand, sondern in einem „Schwallbad" verlötet, indem sie rasch über eine Wanne mit flüssigem Zinn geführt werden. Eine solche Fertigung muß sehr genau einjustiert sein — fährt man die Platine zu schnell über das Zinnbad, so entstehen die gefürchteten „kalten" Lötstellen, die keine einwandfreie Kontaktierung ergeben und sich oft erst nach monatelangem Betrieb durch Ausfälle bemerkbar machen. Dauert der Lötzyklus dagegen zu lange, so können durch die große Hitze Bauelemente beschädigt werden; insbesondere Halbleiter, Kunststoff-Kondensatoren und Elkos sind dagegen allergisch.

Noch ein Wort zur Leiterbahnführung auf Platinen: Es ist auf jeden Fall ratsam, die Bahnen für Masse- und Versorgungsspannungsleitungen etwas breiter zu gestalten (min. etwa 2...3 mm), da sonst einerseits unvertretbar hohe Spannungsabfälle und andererseits eine merkliche Erwärmung die Folge sein können. Leiterbahnen, die hochfrequente Signale bzw. steile Impulse führen, sollte man nicht in geringem Abstand voneinander über die halbe Platine führen — gegenseitige kapazitive Störeinkopplungen wären die Folge. Eine gute Methode ist es, bei doppelseitigen Platinen die Leiterbahnen auf der Oberseite überwiegend längs und auf der Unterseite überwiegend quer zur Platinen-Längsseite zu führen; kapazitive Kopplungen zwischen beiden Platinenseiten werden so minimiert.

Bei sehr komplexen Schaltungen kommt man sogar mit zwei Platinenseiten nicht mehr aus. In diesen Fällen ist man gezwungen, auf die sogenannte Multilayer-Technik überzugehen, bei der die Platinen aus mehreren Leiterbahn- und Isolierschichten aufgebaut sind. Die Herstellung erfolgt durch Übereinanderkleben mehrerer dünner Einzelplatinen mit nachfolgender Durchkontaktierung. Es ist leicht einzusehen, daß so etwas nicht gerade billig ist und daher nur in Ausnahmefällen eingesetzt wird.

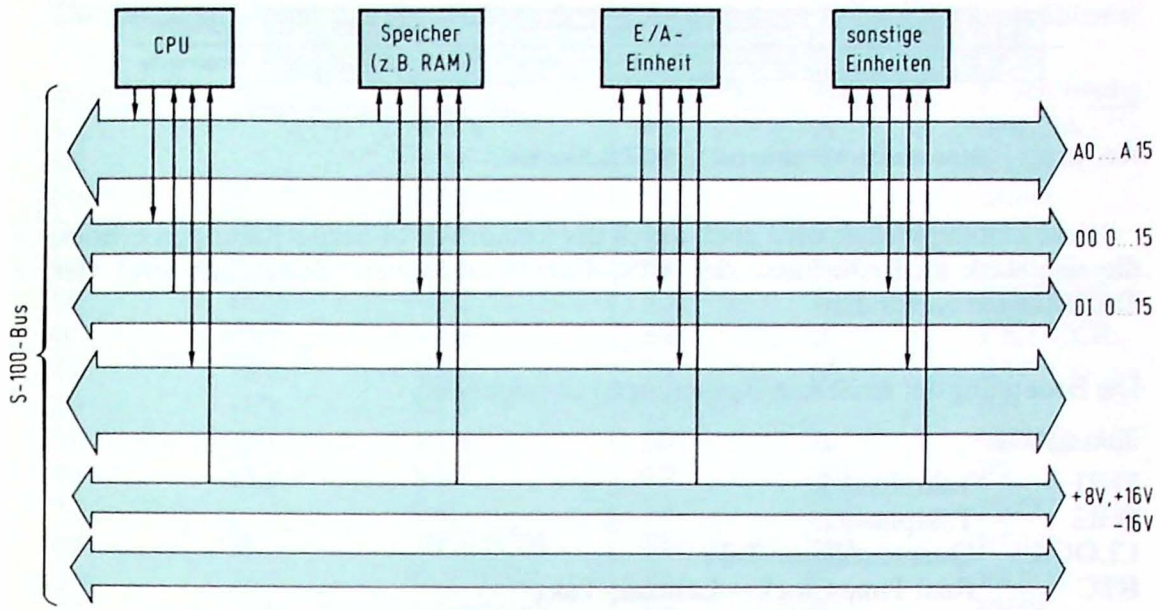


Abb. 1.6.3.1 Der S-100-Bus war lange Zeit ein Quasi-Standard

1.6.3 Busverdrahtung

Wenn hier von Busverdrahtung die Rede ist, dann ist damit der Adressen-, Daten- und Steuerbus innerhalb eines Mikrocomputers gemeint. Auf andere Busarten, die der Verbindung mehrerer Computer untereinander sowie mit Peripheriegeräten dienen, wird im Kapitel „Schnittstellen" noch eingegangen.

Der S-100-Bus

Der wohl älteste und insbesondere in den USA noch sehr verbreitete Bus ist der S-100-Standard. Er wurde 1975 von der Firma MITS erstmals auf einem Mikrocomputerbausatz namens Altair verwendet, mit rund 400 Dollar wohl der erste allgemein erschwingliche Computer.

Der Bus dient dazu, mehrere Baugruppen, von denen jede auf einer eigenen Platine untergebracht ist, miteinander zu verbinden *(Abb. 1.6.3.1)*. Der Name S-100 bedeutet, daß dafür insgesamt 100 Leitungen zur Verfügung stehen, die über einen Platinen-Stecker mit beidseitig je 50 Kontakten hrrausgeführt sind *(Abb. 1.6.3.2)*. Ein wesentliches Merkmal des S-100-Busses ist, daß alle Leitungen Signale nur in einer Richtung führen, also unidirektional ausgelegt sind. Das bedingt die Aufspaltung der acht bidirektionalen CPU-Datenleitungen in 16 (DI 0...7, DO 0...7). Dem Vorteil der unkritischen Auslegung der Bustreiber- und Steuerschaltungen steht der Nachteil der vielen Leitungen gegenüber.

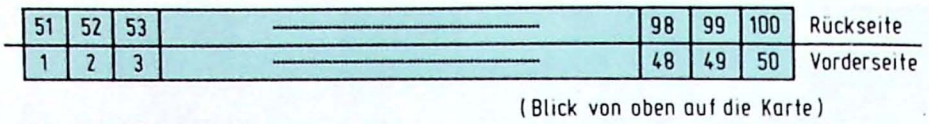


Abb. 1.6.3.2 Zählweise der Kontakte bei S-100-Steckkarten

Diese Leitungsvielfalt wird noch durch die immerhin 44 Steuerleitungen erhöht, die sich stark an Bedürfnisse der 8080-Familie anlehnen. Zusätzlich sind vier Taktleitungen vorhanden.

Die Bedeutung der einzelnen Busleitungen ist folgende:

Taktsignale:

PHI1	Taktphase 1
PHI2	Taktphase 2
CLOCK	Quarzoszillator-Takt
RTC	Real Time Clock = Echtzeit-Takt

Prozessoreingänge:

PRDY	Prozessor bereit
XRDY	Externe Einheit bereit
PHOLD-	Hold-Anforderung
PINT-	Interrupt-Anforderung

Steuersignale:

PINTE	Prozessor für Interrupt bereit
PDBIN	Datenbus frei für externe Einheiten
PWR-	Prozessor-Ausgangsdaten stabil
PSYNC	Beginn eines Maschinenzyklus
PHLDA	CPU ist angehalten (Hold)
PWAIT	CPU ist in einem Wartezyklus

Statusleitungen:

SINTA	Interrupt-Anforderung empfangen
SWO-	Schreiboperation wird ausgeführt
SSTACK	Stackpointer liegt auf Adressenbus
SHLTA	Halt-Befehl wurde empfangen
SOUT	Adresse einer Ausgabe-Einheit auf Bus
SM1	Beginn eines Befehlszyklus
SINP	Adresse einer Eingabe-Einheit auf Bus
SMEMR	Speicher-Lesezyklus
SS	Auf Einzelschritt-Betrieb schalten
RUN	Programm wird abgearbeitet

Die folgende Übersicht zeigt die Signalzuordnung zu den Platinen-Steckanschlüssen:

Anschluß	Richtung	Bezeichnung	Anschluß	Richtung	Bezeichnung
01		+8 V	48	A	SHLTA
02		+16 V	49	A	CLOCK
03	E	XRDY	50		Masse
04	(E)	VI 0	51		+8 V
05	(E)	VI 1	52		−16 V
06	(E)	VI 2	53	E	$\overline{\text{SSW DSB}}$
07	(E)	VI 3	54	A	$\overline{\text{EXT CLR}}$
08	(E)	VI 4	55	(A)	RTC
09	(E)	VI 5	56...67		frei
10	(E)	VI 6	68	A	MWRT
11	(E)	VI 7	69	(E)	$\overline{\text{PS}}$
12...17		frei	70	(A)	PROT
18	E	$\overline{\text{STA DSB}}$	71	(A)	RUN
19	E	$\overline{\text{C/C DSB}}$	72	E	PRDY
20	(A)	UNPROT	73	E	$\overline{\text{PINT}}$
21	(A)	SS	74	E	$\overline{\text{PHOLD}}$
22	E	ADD DSB	75	E	$\overline{\text{PRESET}}$
23	E	DO DSB	76	A	PSYNC
24	A	Φ 2	77	A	$\overline{\text{PWR}}$
25		Φ 1	78	A	PDBIN
26	A	PHLDA	79	A	A0
27	A	PWAIT	80	A	A1
28	A	PINTE	81	A	A2
29	A	A5	82	A	A6
30	A	A4	83	A	A7
31	A	A3	84	A	A8
32	A	A15	85	A	A13
33	A	A12	86	A	A14
34	A	A9	87	A	A11
35	A	DO 1	88	A	DO 2
36	A	DO 0	89	A	DO 3
37	A	A10	90	A	DO 7
38	A	DO 4	91	E	DI 4
39	A	DO 5	92	E	DI 5
40	A	DO 6	93	E	DI 6
41	E	DI 2	94	E	DI 1
42	E	DI 3	95	E	DI 0
43	E	DI 7	96	A	SINTA
44	A	SM1	97	A	$\overline{\text{SWO}}$
45	A	SOUT	98	A	SSTACK
46	A	SINP	99	E	$\overline{\text{POC}}$
47	A	SMEMR	100		Masse

Speichersteuerung:

MWRT	Schreibzyklus des Prozessors
PROT	Schreibschutz-Leitung
UNPROT	Schreibschutz aufheben
PS-	Speicherbereich ist schreibgeschützt

Externer Zugriff:

STADSB-	Statuspuffer auf CPU-Karte ausschalten
C/CDSB-	Befehls- und Steuerpuffer auf CPU-Karte ausschalten
ADDDSB-	Adressenpuffer auf CPU-Karte ausschalten
DODSB-	Datenausgangspuffer auf CPU-Karte ausschalten
SSWDSB-	Dateneingangspuffer auf CPU-Karte ausschalten

Rücksetzleitungen:

POC-	Power-on clear, Einschalt-Initialisierung
PRESET-	Rücksetzsignal für den Prozessor u. a. Logik
EXTCLR-	Impuls zum Löschen von E/A-Einheiten

Die Richtung (E = Eingang, A = Ausgang) ist dabei jeweils von der CPU-Karte aus gesehen. Die eingeklammerten Signale werden nur bei speziellen Anwendungen auf die Karte geführt.

Alle signalführenden S-100-Leitungen haben TTL-Pegel. Die meisten können über Puffer-ICs auf den Karten bis zu 10 Standard-TTL- bzw. 30 Low-Power-Schottky-Eingänge treiben. Die Eingänge der Karten müssen so ausgelegt sein, daß sie maximal einer TTL-Last entsprechen. Auch die direkt vom Prozessor kommenden Signale werden auf der CPU-Karte gepuffert (INTE, DBIN, WR-, SYNC, HLDA, WAIT).

Der optionale Schreibschutz auf einer Speicherkarte muß mit einem Flipflop realisiert werden, das von PROT gesetzt und von UNPROT rückgesetzt wird. Damit ist es möglich, bestimmte RAM-Bereiche vor dem versehentlichen Überschreiben zu schützen.

Das ECB-System

In Europa konnte sich der S-100-Bus aus einem ganz einfachen Grund nie so recht durchsetzen: Während man in Amerika mit ziemlich „wilden" Platinen-Formaten arbeitete, begann sich auf dem europäischen Kontinent schon früh das „Europaformat" (100 × 160 mm) durchzusetzen, wobei auf einer Platinen-Schmalseite eine „richtige" Steckverbindung statt eines nur aus Leiterbahnen bestehenden Platinensteckers üblich war (und ist).

Darüber hinaus begann die Mikrocomputerwelle in Europa erst richtig zu rollen, als der 8080 durch die Einführung des Z80 mit seinen einfacheren Steuer-

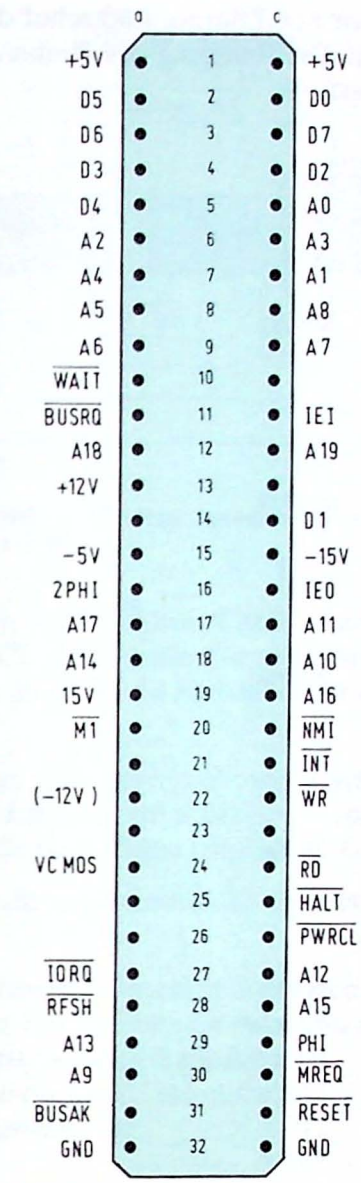


Abb. 1.6.3.3
Belegung des besonders bei Z80-Systemen verbreiteten ECB-Steckers

signalen überholt war. Das Resultat hieß „ECB" (Europe Card Bus), zunächst von Kontron vorgestellt und später von anderen Firmen für Z80-Rechner übernommen. *Abb. 1.6.3.3* zeigt seine Belegung. Er kommt mit insgesamt 64 Anschlüssen einer VG-Steckleiste (DIN 41612) aus. Die Steuersignale orientieren sich verständlicherweise stark an den Bedürfnissen der CPU Z80. Auch hier wird mit TTL-Pegeln gearbeitet.

Der RM-65-Bus

Einer der ersten amerikanischen Hersteller, der sich ebenfalls für das Europakarten-Format entschied, war Rockwell. Diese Firma brachte eine Reihe von Karten

um den Prozessor 6502 heraus und schuf damit eine De-facto-Busnorm für diese Prozessorfamilie. Die Belegung der Reihen a und c der 64poligen VG-Leiste ist beim RM-65-Bus:

	a	c
1	Masse	+5 V
2	—	A15
3	Masse	A14
4	A13	A12
5	A11	Masse
6	A10	A9
7	A8	A7
8	Masse	A6
9	A5	A4
10	A3	Masse
11	A2	A1
12	A0	Phi1
13	Masse	—
14	—	—
15	Ready	Masse
16	—	—
17	—	—
18	Masse	—
19	—	—
20	—	Masse
21	R/W-	—
22	—	—
23	Masse	—
24	IRQ-	NMI-
25	Phi2	Masse
26	—	Reset-Ausg.
27	D7	D6
28	Masse	D5
29	D4	D3
30	D2	Masse
31	D1	D0
32	+5 V	Masse

Alle bisher genannten Bussysteme wurden für 8-Bit-Prozessoren entwickelt und besitzen auch nur 16 Adressenleitungen, so daß (ohne zusätzliche Bank-Switching-Kniffe) maximal 64 KByte Speicher adressiert werden können.

Der Multibus-II

Für komplexere Systeme mit einem oder mehreren 16-Bit-Prozessoren haben sich mehrere Hersteller unter Leitung von Intel gemeinsam ein Konzept überlegt, das

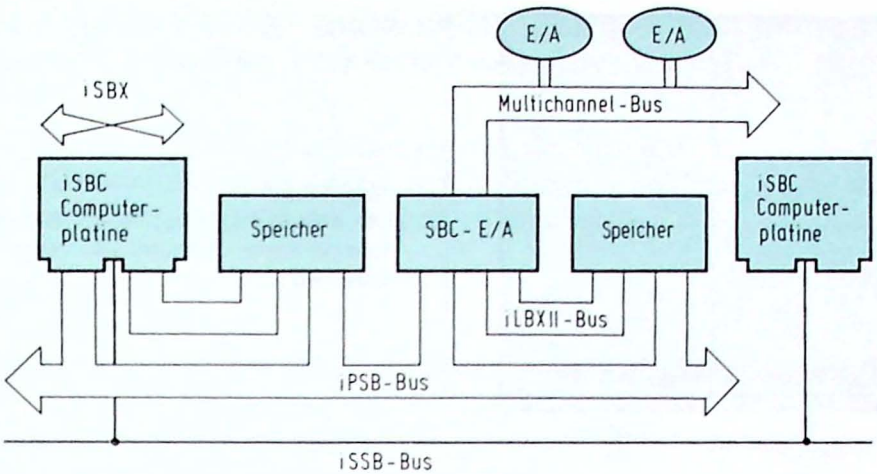


Abb. 1.6.3.4 Das Multibus-Konzept von Intel: iPSB = paralleler Systembus, iLBX II = lokaler Bus, iSSB = serieller Systembus, Multichannel = DMA-I/O-Bus

als Multibus-II Verbreitung fand: AEG, AMD, Altos, CMB, Dataindustries AB, Ericsson, Foxboro, Hewlett-Packard, ICL, Intel, Intersil und das Kernforschungszentrum Jülich. Inzwischen haben sich mehr als 50 Hersteller für diese Busnorm entschieden.

Wie der Name schon sagt, handelt es sich dabei nicht um einen einzigen Bus. Vielmehr geht das Konzept davon aus, daß mehrere Busteile die unterschiedlichen Anforderungen der einzelnen Systemteile befriedigen *(Abb. 1.6.3.4)*.

Der wichtigste Teil ist dabei der parallele Systembus. Er besteht aus folgenden Leitungen:

AD 0...31	32 Adressen-/Datenleitungen (gemultiplext)
PAR 0...3	Paritäts-Kontrollbits für Adressen und Daten
SC 0...8	Systemsteuerungs-Leitungen
BUSERR, TIMOUT, NACK	Statusleitungen (Exceptions)
RST, RSTNC, DCLOW, PROT, BCLK, CCLK	Zentralsteuerung
BREQ, ARB 0...5	Arbitration
+5 V, +5-V-Batterie, +12 V, −12 V, Masse	Stromversorgung

Unter „Arbitration" versteht man die Freigabe des Bussystems für eine Karte, bei einem System aus mehreren Prozessoren also für einen Prozessor. Die übrigen Prozessoren können dann auf ihrem lokalen Bus weiterarbeiten, der gemeinsame Systembus ist für den Datentransfer zwischen zwei vom „Arbiter" (Buszuordnungs-Baustein) bestimmten Systemkomponenten reserviert. Die Arbitration erfolgt dezentral. Mit BREQ meldet eine CPU, daß sie auf den Bus zugreifen möchte. Wenn mehrere CPUs zur selben Zeit zugreifen wollen, sorgt eine Prioritäts-Steuerung im Arbiter für die korrekte Reihenfolge der Buszuweisung.

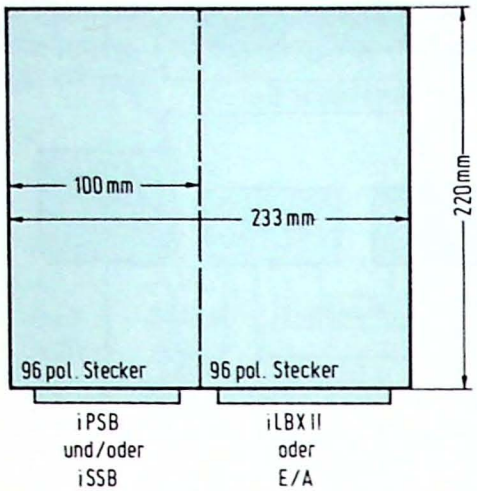


Abb. 1.6.3.5
Der Intel-Multibus verwendet Doppel-Europa-Karten mit zwei 96poligen Steckern

Es wird mit 32 Bit breiten Speicheradressen gearbeitet, das entspricht einer maximalen Adressierbarkeit von 4 GByte. Für den Ein-/Ausgaberaum stehen 16 Adressenleitungen zur Verfügung, so daß max. 65536 E/A-Ports ansprechbar sind.

Die Steuerleitungen haben folgende Aufgaben:

SC0	Buszyklus-Anfang (Adresse wird angelegt)
SC1	Bus für andere CPUs sperren
SC2...3	Datentransfer 8/16/24/32 Bit breit
SC4...5	Selektion Speicher, E/A, Interconnect, Message-Adreßraum
SC6	Read/Write (Lesen/Schreiben)
SC7	Fehler-Zurückmeldung (mit SC5 und SC6)

Während eines Datentransfers kann derjenige, der den Transfer startet, die Daten verzögern, mit SC4 derjenige, der angesprochen wird. So kann die CPU bei einem Speicher-Schreibvorgang mit SC3 Wartezyklen einfügen und sich einem schnellen Speicher anpassen; mit SC4 kann umgekehrt der Speicher Wartezyklen einfügen und sich einer schnelleren CPU anpassen.

Eine Besonderheit des Multibus-Konzepts ist der serielle Systembus ISSB. Er ermöglicht die Verbindung von Platinen über eine Zweidraht-Leitung. Das ist besonders zweckmäßig, wenn diese Platinen räumlich weit voneinander getrennt sind. Dabei wird folgende Konvention getroffen:

1. Jede Station „hört" auf der Leitung mit.
2. Solange die Leitung belegt ist, muß man warten.
3. Sobald die Leitung frei ist, darf eine Station senden.
4. Beginnen zwei Stationen gleichzeitig zu senden (Kollision), so brechen zunächst alle Stationen ab. Nach einer Zufallszeit beginnt eine Station wieder zu senden.

Natürlich funktioniert der serielle Datentransfer nicht ganz so schnell wie auf dem parallelen Systembus; dafür kommt man aber, wie gesagt, mit nur zwei Leitungen aus.

Der Multibus-II-Standard arbeitet mit Doppel-Europakarten (IEC-297-Norm) mit zwei 96poligen Steckern *(Abb. 1.6.3.5)*. Nähere Unterlagen hierzu erhält man auf Anforderung von einer der oben genannten Firmen.

Der VME-Bus

Während der Multibus-Standard von seinen Steuersignalen her in erster Linie für die 16- und 32-Bit-Prozessoren von Intel ausgelegt ist, haben sich Motorola, Valvo/Signetics und andere Hersteller des Prozessors 68000 zu einem eigenen 16-Bit-Busstandard namens VME-Bus entschlossen.

Der VME-Bus geht wie der Multibus von Doppel-Europakarten aus, die allerdings nicht mit zwei, sondern nur mit einer 96poligen Steckleiste als Verbindung zwischen den einzelnen Baugruppen ausgerüstet sind. Daraus ergibt sich schon, daß der VME-Bus mit weniger Leitungen auskommt.

Trotzdem handelt es sich um einen echten 16-Bit-Bus mit 16 Daten- und 24 Adressenleitungen, so daß sich 16 MByte adressieren lassen. Der Datentransfer ist asynchron und bidirektional möglich, auch auf Steuerleitungen. Dadurch kann ein VME-System auch mit mehreren CPU-Karten aufgebaut sein, die gegebenenfalls über die Bus-Request-Steuerleitung den Bus für sich beanspruchen. Die übrigen CPUs können dann auf ihrem karteninternen „Privatbus" weiterarbeiten.

Die einzelnen Karten werden auf eine gemeinsame Mutterplatine (Backplane, Motherboard) gesteckt, deren Steckleisten man als „Slots" bezeichnet. Der System-Controller, also jene CPU-Karte mit der höchsten Priorität, muß (!) dabei in Slot 1 stecken (daisy chain arbitration).

Für genauere Informationen über diesen schon wegen des asynchronen Betriebs recht komplexen Bus, der eine Besonderheit der CPU 68000 darstellt, wird auf die äußerst umfangreiche Spezifikation von Motorola bzw. Valvo hingewiesen.

Herstellerspezifische Bussysteme

In Tischcomputern werden meist keine standardisierten, sondern herstellerspezifische Bussysteme eingesetzt. Typisch dafür sind IBM-PC und Apple-II: Die CPU sitzt zusammen mit einer Speicher- und I/O-Minimalausstattung auf einer großen Mutterplatine, die für spätere Erweiterungen sogenannte Slots enthält. Das sind Steckleisten, in die man Karten für unterschiedlichste Anwendungen stecken kann. Sowohl beim Apple-II als auch beim IBM-PC tragen diese Karten keine richtigen Steckkontakte: Vielmehr dienen an den Rand geführte Leiterbahnen als Verbindung zur Steckleiste der Mutterplatine. Bei beiden Computern umfaßt der Datenbus acht Leitungen, da beide 8-Bit-CPUs verwenden (6502 bzw. 8088).

Beim Apple-II haben die 50 Anschlüsse eines der acht Slots folgende Bedeutung:

Pin	Name	Bedeutung
1	I/O Select-	Low, wenn Adr. = CnXX mit n = Slot-Nr.
2—17	A 0...15	Gepufferter Adressenbus
18	R/W-	Read/Write (wird gleichz. m. Adr.-Bus gültig)
19	SYNC	Video-Synchronsignal, nur bei Slot 7
20	I/O-Strobe-	Low, wenn Adresse = C800...CFFF
21	RDY	Ready-Eingang der 6502-CPU
22	DMA-	Mit Low-Signal wird die 6502-CPU angehalten
23	INT OUT	Daisy-Chain-Interrupt-Leitung (s. Pin 28)
24	DMA OUT	Daisy-Chain-DMA-Ltg., meist mit Pin 27 verb.
25	+5 V	Max. 500 mA für alle Slots zusammen
26	GND	Masse, Bezugspotential
27	DMA IN	Daisy-Chain-DMA-Eingang (s. Pin 24)
28	INT IN	Daisy-Chain-Interrupt-Eingang (s. Pin 23)
29	NMI-	Low = nicht maskierbarer Interrupt
30	IRQ-	Low = Interrupt-Request
31	RES-	Low = Reset
32	INH-	Bei Low werden alle ROMs abgeschaltet
33	−12 V	Max. 200 mA für alle Slots zusammen
34	−5 V	Max. 200 mA für alle Slots zusammen
35	COLOR REF	Farbträgersignal (nur bei Slot 7)
36	7 M	7-MHz-Takt
37	Q 3	Asymmetrischer 2-MHz-Takt
38	PHI1	Prozessor-Takt (1,023 MHz)
39	USER1	Bei Low werden die internen I/O-Adressen abgeschaltet
40	PHI0	Prozessor-Takt (1,023 MHz)
41	DEV SEL-	Low bei Adr. = C0nX mit n = Slot-Nr.+8
42—49	D 7...0	Gepufferter, bidirektionaler Datenbus
50	+12 V	Max. 250 mA für alle Slots zusammen

Alle Signale haben dabei TTL-Pegel. Interessant ist die Möglichkeit, mit DMA den 6502-Prozessor anzuhalten und mit INH die ROM-ICs auf der Mutterplatine abzuschalten. Dadurch ist es möglich, in einen der Apple-Slots eine Karte mit einem anderen Prozessor (z.B. Z80 oder 68008) zu stecken. Der durch das ROM-Abschalten gewonnene Adressenraum kann mit zusätzlichem RAM gefüllt werden und somit z.B. das Betriebsprogramm für die „fremde" CPU aufnehmen, ohne daß ein Teil des übrigen Arbeitsspeichers verloren geht *(Abb. 1.6.3.6)*.

Es würde zu weit führen, auf alle herstellerspezifischen Bussysteme einzugehen. Das Slot-System des Apple-II wurde jedoch von zahlreichen Anbietern kompatibler Geräte übernommen, und viele Drittfirmen bieten dafür Erweiterungskarten für unterschiedlichste Anwendungen an.

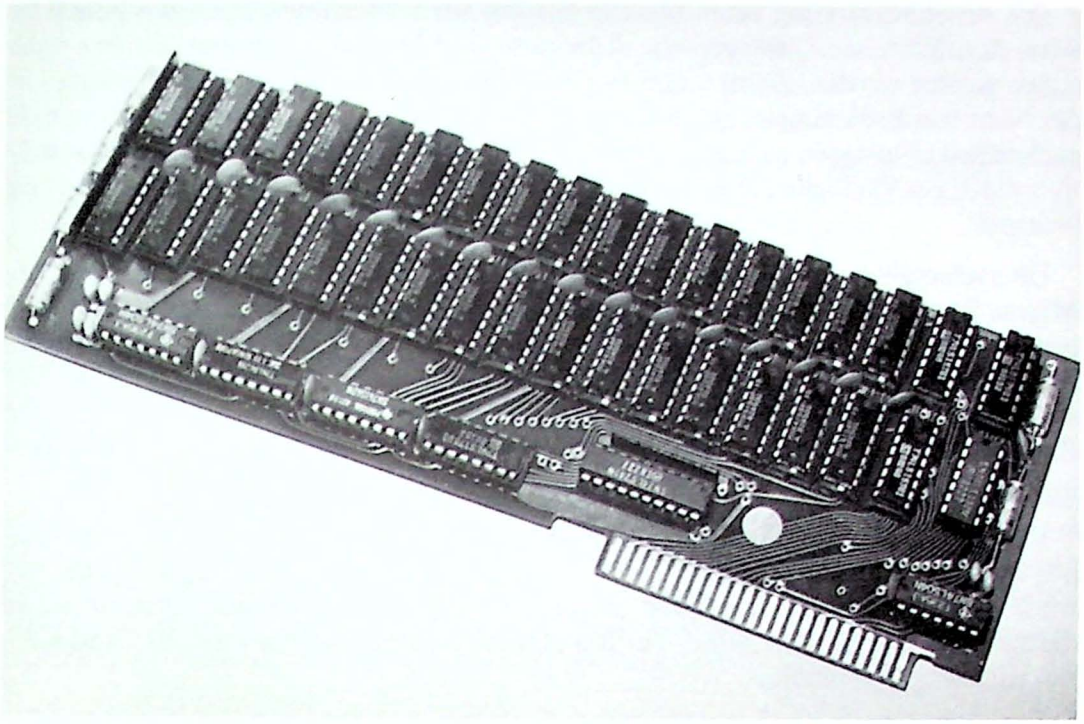


Abb. 1.6.3.6 Speicherkarte für den Apple-II-Bus

1.6.4 Gehäuse, Störstrahlung, Wärmeabfuhr

Als Gehäuse für Mikrocomputer stehen heute Ausführungen unterschiedlichster Abmessungen aus industrieller Serienproduktion zur Verfügung. Wenn der Rechner mit einem externen Terminal betrieben werden soll, liegt bei Einzelanfertigungen die Verwendung eines quaderförmigen Gehäuses nahe. Aber auch für Geräte, bei denen die Tastatur und vielleicht auch ein Bildschirm eingebaut sind, gibt es geeignete Ausführungen entsprechender Formgebung. In der Industrie sind sogenannte 19-Zoll-Einschübe sehr verbreitet; sie besitzen eine Breite von 19 Zoll und eignen sich besonders gut zur Aufnahme von Doppel- oder Einfach-Europakarten (je nach Bauhöhe).

Eine wesentliche Frage ist, aus welchem Material das Gehäuse beschaffen sein soll. Kunststoff bietet meist preisliche Vorteile und ist relativ leicht. Metallgehäuse (vorzugsweise aus dem leichten und einfach bearbeitbaren Aluminium hergestellt) besitzen dagegen den Vorteil höherer mechanischer Stabilität und vermindern zudem erheblich die Störstrahlung, die der Computer in seiner näheren Umgebung in einem breiten Frequenzband verursacht (mit einem daneben gestellten Kofferradio ist sie leicht feststellbar). Umgekehrt schirmt ein Metallgehäuse auch den Computer gegen von außen kommende Störimpulse ab (z.B. elektromagnetisches Feld bei einem Blitzeinschlag in der Nähe).

Die Abschirmwirkung eines Metallgehäuses wird allerdings zunichte gemacht, wenn signalführende Leitungen wie Adressen- und Datenbus unabgeschirmt nach außen geführt werden. Selbst wenn nur das Netzkabel innerhalb des Gehäuses in der Nähe von Busleitungen geführt wird, kann es deren hochfrequente Störsignale nach außen übertragen und umgekehrt externe Störimpulse aus dem Netz kapazitiv auf den empfindlichen Bus leiten. Das ist bei der Leitungsführung unbedingt zu beachten.

Überschreitet ein Gerät (das gilt nicht nur für Computer) die von der Post als Maximalwert zugelassene Störstrahlungs-Feldstärke und tritt tatsächlich eine Störung von Fernmeldeanlagen der Nachbarschaft auf (Rundfunk, Fernsehen, Funkdienste), so kann bis zur Beseitigung der Störung die Einstellung des Betriebs dieses Computer verfügt werden. Es ist also sehr empfehlenswert, sich schon in der Entwicklungsphase auch Gedanken darüber zu machen, wie die nach außen dringende Störstrahlung möglichst gering gehalten werden kann.

Es ist weder sinnvoll noch für die Abschirmwirkung nötig, das Gehäuse sozusagen luftdicht zu gestalten. Vielmehr produziert praktisch jeder Computer eine gewisse Verlustwärme, die irgendwie an die Luft abgegeben werden muß, da sonst die zulässige Betriebstemperatur der Bauelemente (meist weniger als 70 Grad Celsius) überschritten würde.

Je nach Verhältnis von Verlustwärme und Gehäusegröße genügt oft das Anbringen von Lüftungsschlitzen, vorzugsweise auf der Ober- und Unterseite, da die Luft dann unten eintreten und nach Erwärmung oben austreten kann. Dieses Verfahren heißt Konvektionskühlung und nützt die Tatsache aus, daß die erwärmte Luft ein geringeres spezifisches Gewicht besitzt und sich deshalb nach oben bewegt.

Die Alternative bei größerer Verlustwärme ist der Einbau eines kleinen Ventilators. Solche „Lüfter" gibt es als Radial- oder Tangential-Ausführungen; letztere eignen sich besonders für flache Gehäuseformen. Die Stromversorgung erfolgt entweder mit der Netzwechselspannung oder mit der im Computer-Netzteil erzeugten Niederspannung.

Ventilatoren sollte man allerdings nur dann einsetzen, wenn eine Konvektionskühlung nicht ausreichend möglich ist. Sie verursachen immer (!) ein Geräusch, das bei ruhiger Umgebung recht störend sein kann. Es ist daher ratsam, Lüfter resonanzfrei z.B. auf Gummipuffern einzubauen, wenn auf sie nicht verzichtet werden kann.

1.7 Verbindung nach draußen

1.7.1 Tastaturen

Wie man Tastaturen mit dem Computer-Bus verbindet und wie sie decodiert werden, wurde schon im Kapitel über Peripheriebausteine geklärt. Hier soll es in erster Linie um die mechanische und elektrische Ausführung gehen.

Bei preiswerten Heimcomputern *(Abb. 1.7.1.1)* findet man oft einfache Drucktasten-Elemente, die an Taschenrechner-Tasten erinnern. Auch wenn die Buchstaben in ähnlicher Form wie bei Schreibmaschinen angeordnet sind, sind sie für eine rasche, bequeme Texteingabe nicht geeignet (und wohl auch nicht gedacht).

Professionelle Tastaturen *(Abb. 1.7.1.2)* sind andererseits natürlich teurer. Abgesehen von der sorgfältigeren Auslegung der einzelnen Tastenelemente ist neben dem alphanumerischen Block oft ein getrennter Ziffernblock sowie manchmal auch eine Reihe von Funktionstasten vorhanden, die sich mit kompletten Befehlswörtern belegen lassen, um beim Programmieren Zeit zu sparen. In solchen komfortablen Tastaturen ist meist ein eigener Mikroprozessor eingebaut.

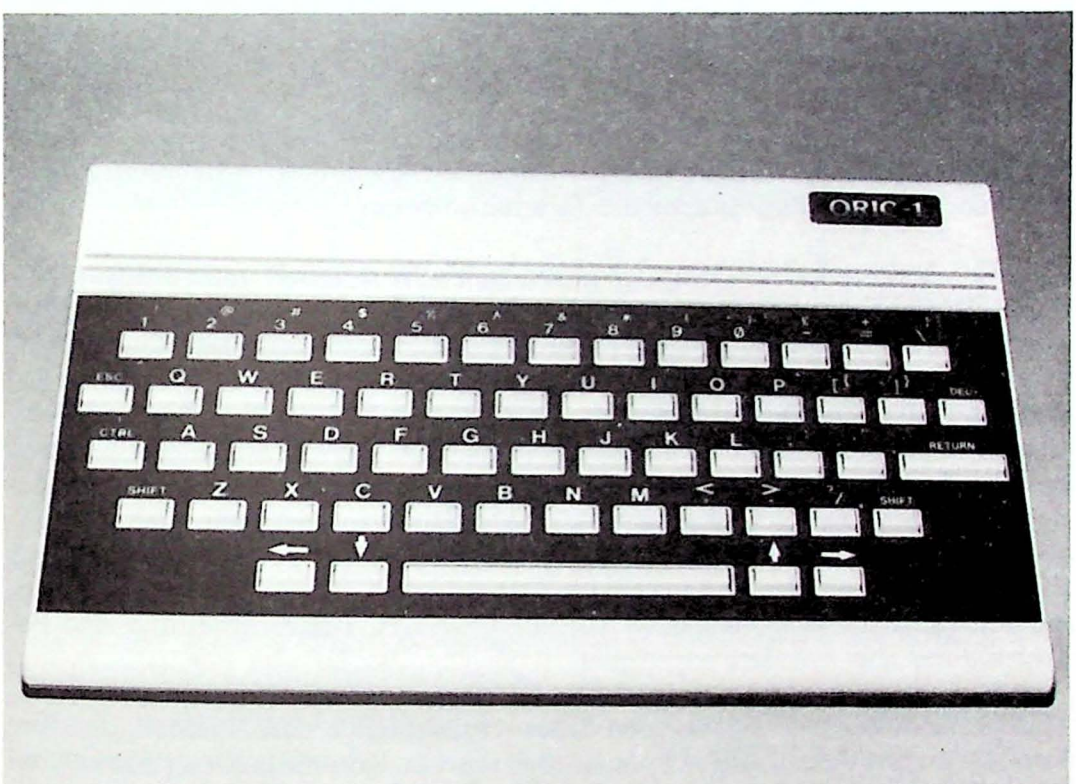


Abb. 1.7.1.1 Heimcomputer mit vereinfachter Tastatur (Oric-1)

Abb. 1.7.1.2 Komfortable Tastatur mit getrennten Cursor- und Ziffern-Feldern (AFC)

Neben den herkömmlichen elektromechanischen Tastenelementen, bei denen einfach zwei federnde Kontakte überbrückt werden, sobald die Taste betätigt wird, gibt es noch weitere Möglichkeiten der Tastenausführung.

Die sogenannten Folien-Tastaturen lassen sich sehr rationell und preiswert herstellen. Darüber hinaus haben sie den Vorteil, daß ihre Betätigungsfläche gegen verschiedenste Umwelt-Einflüsse hermetisch dicht ist. Eine leitfähige Gummifolie überbrückt bei Betätigung zwei darunterliegende Kontakte, die z.B. auf einer gedruckten Leiterplatte einfach herstellbar sind *(Abb. 1.7.1.3)*. Auf der Oberseite der Gummischicht kann man zusätzlich eine bedruckte Isolierfolie anbringen. Nachteilig ist bei der Folientastatur der sehr geringe „Tastenhub", also der Weg, der bei der Betätigung vertikal zurückgelegt wird. Man hat also sozusagen kein Gefühl dafür, ob die Taste nun richtig gedrückt wurde oder nicht. Eine gewisse Verbesserung läßt sich dadurch erzielen, daß man jeden Tastendruck z.B. mit einem kurzen Piepton bestätigt. Für eingabe-intensive Aufgaben wie z.B. Textverarbeitung sind Folientastaturen aber trotzdem ungeeignet.

Mechanisch arbeitende Tastaturen haben grundsätzlich den Nachteil, daß ihre Kontakte mit der Zeit oxidieren können. Das führt zu Kontaktprellen (Wackelkontakt) oder dazu, daß die Taste überhaupt nicht mehr funktioniert. Kapazitive und Hall-Tastaturen lösen dieses Problem, haben allerdings aber auch ihren Preis.

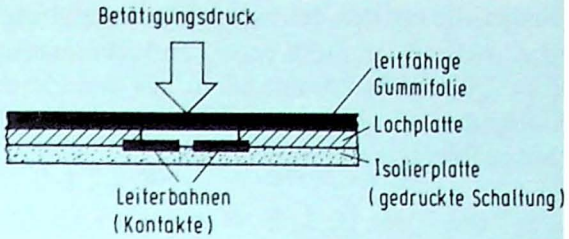


Abb. 1.7.1.3
Aufbau einer Folien-Taste

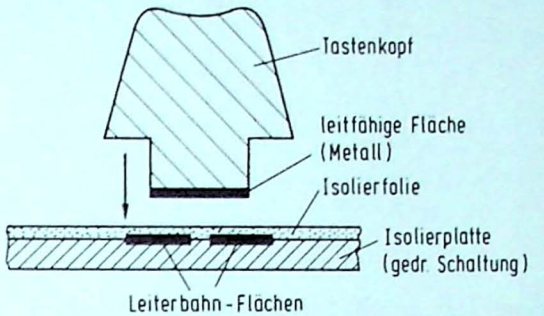


Abb. 1.7.1.4
Prinzip einer kapazitiv arbeitenden Taste

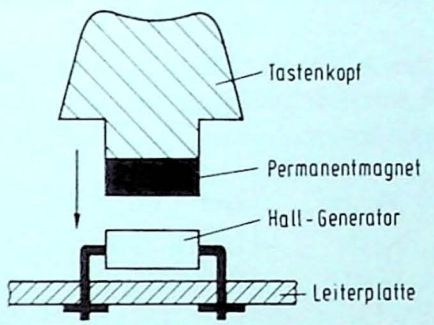


Abb. 1.7.1.5
Kontaktlose Taste mit Hall-Generator

Eine kapazitive Tastatur *(Abb. 1.7.1.4)* besteht aus einer gedruckten Schaltung (Leiterplatte), die mit einer dünnen Isolierfolie überzogen ist. Der auf seiner Unterseite metallisierte Tastenkopf sorgt bei Betätigung dafür, daß er zusammen mit den Leiterbahn-Flächen wie ein Kondensator wirkt, der für hohe Frequenzen nur einen geringen Widerstand darstellt. Die Decoder-Schaltung für eine solche kapazitive Tastatur ist also etwas aufwendiger als für eine mechanische.

Eine weitere Möglichkeit, „Kontaktschwierigkeiten" zu vermeiden, ist die Verwendung eines Hall-Generators (s. Kapitel über Sensoren) in jedem Tastenelement. Der Tastenkopf enthält einen Permanentmagneten, der bei Annäherung an den Hall-Generator dessen Brückenschaltung aus dem Gleichgewicht bringt *(Abb. 1.7.1.5)*. Wegen der geringen Ausgangsspannung sind Verstärkerschaltungen nötig, um auf normale Logik-Pegel zu kommen.

Noch ein Wort zur Anordnung der einzelnen Zeichen auf der Tastatur. Amerikanische Tastaturen besitzen die sogenannte „QWERTY"-Anordnung, d.h. das Y sitzt in der zweiten Reihe rechts neben dem T. Gegenüber dem deutschen Standard sind daher Y und Z vertauscht. Ebenso enthält eine amerikanische Tastatur keine

Tasten, die mit den deutschen Umlauten beschriftet sind. Wenn sie aber eckige und geschweifte Klammern erzeugen kann, erzeugt sie sehr wohl die korrekten ASCII-Zeichen, und es kommt allein auf den Zeichengenerator im Video-Interface des Computers an, ob diese als Klammern oder als Umlaute dargestellt werden. Hierbei gibt es folgende Entsprechungen:

US-Taste	Dt. Umlaut	Hex-Code (ASCII)
[	Ä	5B
\	Ö	5C
]	Ü	5D
{	ä	7B
|	ö	7C
}	ü	7D
~	ß	7E
@	§	40

Diese Tabelle gilt für alle Computer, die intern mit ASCII-Zeichen arbeiten; für MS-DOS-Computer trifft dies allerdings meist nicht zu.

Die meisten ASCII-Tastaturen besitzen auch eine CTRL-Taste (sprich: Control), mit der sich durch gleichzeitiges Betätigen einer Buchstabentaste ASCII-Steuerzeichen erzeugen lassen:

Tasten	Steuerz.	Hex-Code
CTRL-§	NUL	00
CTRL-A	SOH	01
CTRL-B	STX	02
CTRL-C	ETX	03
CTRL-D	EOT	04
CTRL-E	ENQ	05
CTRL-F	ACK	06
CTRL-G	BEL	07
CTRL-H	BS	08
CTRL-I	HT	09
CTRL-J	LF	0A
CTRL-K	VT	0B
CTRL-L	FF	0C
CTRL-M	CR	0D
CTRL-N	SO	0E
CTRL-O	SI	0F
CTRL-P	DLE	10
CTRL-Q	DC1	11
CTRL-R	DC2	12
CTRL-S	DC3	13
CTRL-T	DC4	14

CTRL-U	NAK	15
CTRL-V	SYN	16
CTRL-W	ETB	17
CTRL-X	CAN	18
CTRL-Y	EM	19
CTRL-Z	SUB	1A
CTRL-Ä	ESC	1B
CTRL-Ö	FS	1C
CTRL-Ü	GS	1D
CTRL-↑	RS	1E
CTRL-_	US	1F

Bedingt durch die Auslegung des Tastatur-Decoders sind manchmal allerdings nicht alle dieser Steuerzeichen wirklich erreichbar. Ihre genaue Bedeutung ist im Kapitel 1.1.3 nachzulesen.

1.7.2 Alphanumerische Anzeigen

Im Gegensatz zu Taschenrechnern arbeiten Computer (soweit sie nicht ausschließlich Steuerungszwecken dienen) mit Anzeigen, die nicht nur Ziffern, sondern auch Buchstaben und Satzzeichen wiedergeben können. Solche Displays nennt man „alphanumerisch". Je nach Anwendungsfall kommen dafür sehr unterschiedliche Techniken in Frage.

LED-Segment-Display

Mehrere Leuchtdioden (LEDs) lassen sich in einem gemeinsamen Gehäuse so integrieren, daß sie eine Anordnung von Segmenten bilden, aus denen sich alle nötigen Zeichen kombinieren lassen *(Abb. 1.7.2.1)*. Bei rotleuchtenden LEDs ist die anzulegende Spannung etwa 1,5 V, und pro Segment muß man mit etwa 5...10 mA Strom für eine ausreichende Helligkeit rechnen. Man kann sich leicht überlegen, daß diese Technik daher nur für Anzeigen mit verhältnismäßig wenigen Zeichen in Frage kommt. So kann der Strom bei einer 16-Segment-Anzeige mit nur 20 Zeichen maximal immerhin 20 × 16 × 5 mA = 1,6 A betragen (bei 5 mA pro Segment). Allerdings werden im Normalbetrieb kaum alle Segmente gleichzeitig leuchten.

Solche LED-Displays sind auch in Gruppen von z.B. je vier Zeichen lieferbar und enthalten z.T. auch schon die nötige ASCII-Ansteuer-Elektronik. Wegen der

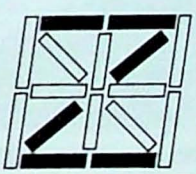


Abb. 1.7.2.1
Darstellung des Buchstabens „Z" auf einem 16-Segment-Display

schlechten Ablesbarkeit bei hoher Umgebungshelligkeit und des schon erwähnten hohen Stromverbrauchs setzt man sie heute aber nur noch vereinzelt ein.

Fluoreszenz-Anzeigen

Auch bei Fluoreszenz-Anzeigen *(Abb. 1.7.2.2)* werden die einzelnen Zeichen aus mehreren Segmenten zusammengesetzt. Solche Anzeigen sind ähnlich wie eine Elektronenröhre aufgebaut; sie arbeiten nämlich mit einem Heizdraht als Katode, und jedes Segment stellt eine Anode dar. Die Anoden sind mit einer fluoreszierenden Schicht versehen. Sie leuchtet blaugrün (manchmal auch gelblich) auf, sobald zwischen Anode und Katode eine Spannung angelegt wird, so daß die von der Katode durch Hitze ausgelösten Elektronen mit genügender Geschwindigkeit auf die Anode aufprallen. Damit sie nicht gebremst werden, ist das Innere der aus Glas bestehenden Anzeigenröhre (sie enthält meist gleich mehrere Zeichen, z.B. 16) luftleer gepumpt.

Der Stromverbrauch von Fluoreszenz-Anzeigen ist wesentlich geringer als der von LED-Displays gleicher Zeichenzahl. Allerdings ist zu ihrem Betrieb eine relativ hohe Spannung (ca. 20...30 V) erforderlich, was auch bei den nötigen Ansteuerschaltungen berücksichtigt werden muß.

Abb. 1.7.2.2 Entwicklungssystem von Rockwell mit 16-Segment-Fluoreszenz-Anzeige

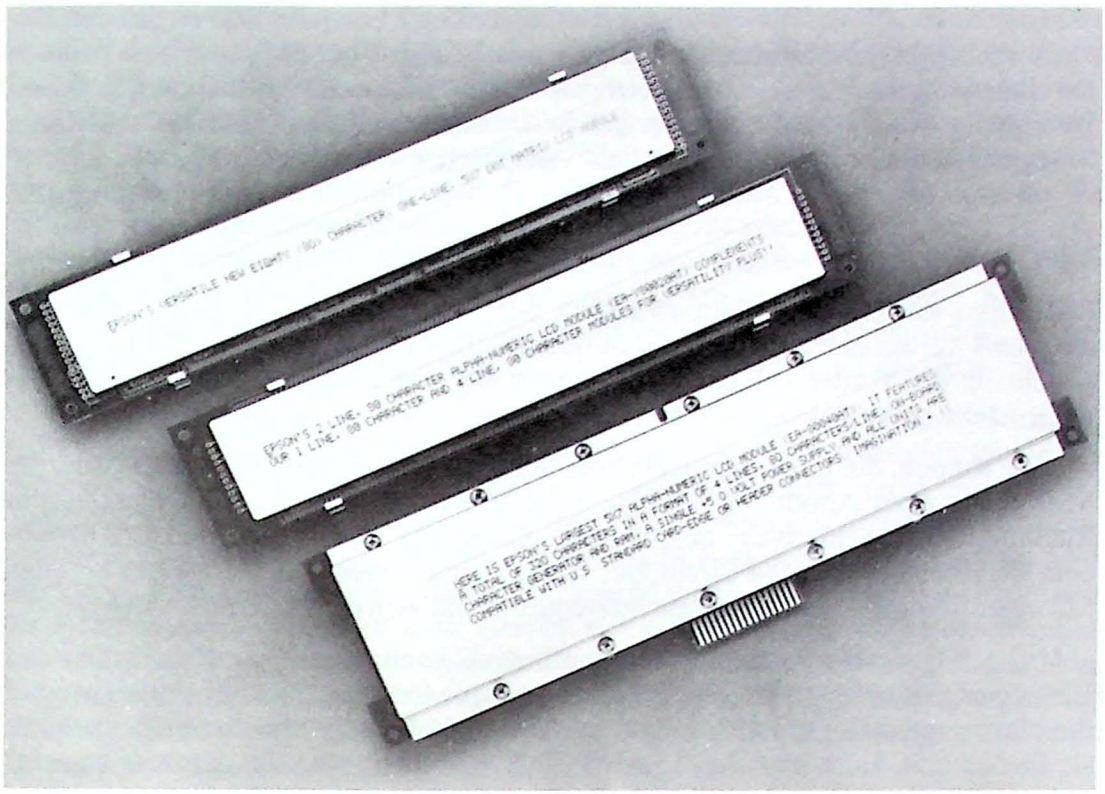


Abb. 1.7.2.3 LCD-Moduln mit 80 Zeichen Breite und 1...4 Zeilen Höhe

Flüssigkristall-Anzeigen (LCDs)

Die Wirkungsweise von LC-Displays wurde schon im Kapitel über Bauelemente beschrieben. Ihr Hauptvorteil ist der sehr geringe Stromverbrauch, und sie lassen sich auch bei Spannungen von 5...12 V betreiben, wie sie in einem Mikrocomputer ohnehin vorhanden sind.

Alphanumerische LC-Anzeigen lassen sich wie Fluoreszenz- und LED-Displays aus Segmenten zusammensetzen; Kleinbuchstaben lassen sich damit allerdings nicht vernünftig darstellen.

Vielfach verwendet man daher ein Punktraster für jedes Zeichen *(Abb. 1.7.2.3)*. Auf diese Weise lassen sich auch größere Flüssigkristall-Displays realisieren, auf denen bis zu 24 Zeilen mit je 80 Zeichen Platz finden (!). Bei Aktentaschen-Computern sind LC-Displays mit acht Zeilen zu je 40 Zeichen verbreitet. Das LCD-Punktraster eignet sich natürlich auch zur Darstellung von Grafiken.

Da LCDs nicht selbstleuchtend sind, sondern nur ihr Licht-Reflektionsverhalten unter dem Einfluß der angelegten Spannung ändern, eignen sie sich auch bei hoher Umgebungshelligkeit, ja sogar bei direkter Sonneneinstrahlung. Allerdings ist ein ausreichender Kontrast nur in einem bestimmten Betrachtungswinkel-Bereich ge-

währleistet. Durch Ändern der angelegten Spannung kann man den optimalen Betrachtungswinkel aber einstellen. Ein gewisser Nachteil bei LC-Anzeigen ist auch der eingeschränkte Temperaturbereich; sie funktionieren nur etwa zwischen 0 und 50 Grad Celsius. Schließlich eignen sich LCDs nicht besonders für die Darstellung bewegter Grafiken, da sie eine merkliche Ansprechzeit besitzen (ca. 0,1 s).

Bildröhren

Bildröhren sind von der Fernsehtechnik her bekannt, und die in der Computertechnik verwendeten sind bis auf die chemische Zusammensetzung der Leuchtschicht auf der Vorderseite ebenso aufgebaut wie ihre Kollegen aus der Unterhaltungs-Elektronik *(Abb. 1.7.2.4)*.

Eine Katode, die mit einem Heizfaden erhitzt wird, sendet Elektronen aus. Sie werden von einer Anode (an der eine hohe Spannung von 10...20 kV anliegt) in Richtung der Leuchtschicht beschleunigt. Geeignete Fokussier-Elektroden bündeln sie zu einem scharfen Strahl, der einen kleinen, hellen Leuchtpunkt erzeugt. Mit einer weiteren Elektrode, dem Steuergitter, läßt sich die Helligkeit steuern.

Um den gesamten Bildschirm beschreiben zu können, sind zwei Gruppen von Elektromagneten außen um den Röhrenhals angeordnet, nämlich die Vertikal- und die Horizontal-Ablenkspulen. Indem man sägezahnförmige Wechselströme durch sie fließen läßt, kann man den Elektronenstrahl zeilenweise die gesamte Leucht-

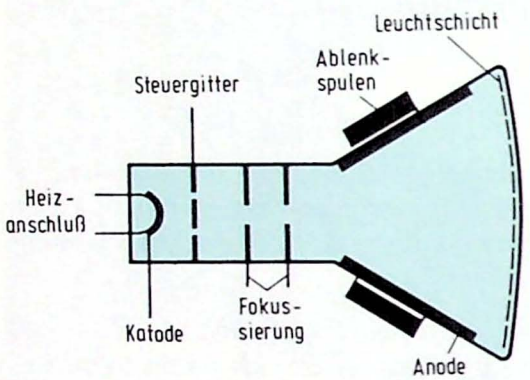


Abb. 1.7.2.4
Aufbau einer Schwarzweiß-Bildröhre

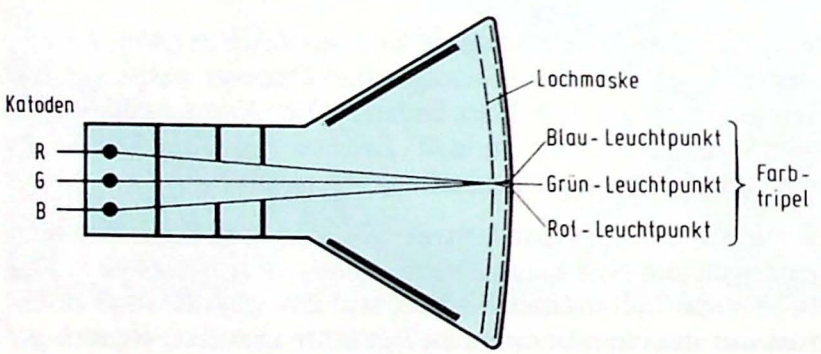


Abb. 1.7.2.5
Aufbau einer Farbbildröhre. Für jede Farbe ist eine Katode vorhanden

schicht beschreiben lassen. Die Darstellung von Grafiken und Schrift erreicht man einfach durch geeignete Helligkeits-Steuerung während des Abtastvorgangs (siehe Kapitel über Video-Interface-Bausteine). Die Darstellungsfarbe hängt im wesentlichen von der chemischen Zusammensetzung der Leuchtschicht ab; in der Datentechnik sind weiße, grüne und gelbliche Phosphore üblich. Die Zusammensetzung der Leuchtschicht bestimmt auch die Nachleuchtdauer; das ist die Zeit, die die Schicht nach dem Abschalten des Elektronenstrahls noch sichtbar leuchtet. Eine lange Nachleuchtdauer, z.B. 0,3 s, vermindert zwar Flimmereffekte, führt aber andererseits auch zu Unschärfen bei bewegten Grafiken („Schmieren"). Flimmern läßt sich aber auch durch eine ausreichend hohe Vertikal-Ablenkfrequenz vermeiden (min. 50 Hz bei Verzicht auf das 25-Hz-Halbbild-Verfahren des Fernsehens).

Bei Farbbildröhren sind drei Katoden vorhanden, je eine für die drei Grundfarben Rot, Grün und Blau, aus denen sich alle anderen Farben durch additive Mischung zusammensetzen lassen. Ebenso ist die Leuchtschicht nicht durchgehend aus einer Farbe hergestellt, sondern enthält dicht nebeneinanderliegend rote, grüne und blaue Punkte. Kurz vor der Leuchtschicht muß der Elektronenstrahl noch eine sogenannte Lochmaske passieren; sie sorgt dafür, daß die von der Blau-Katode kommenden Elektronen auch wirklich nur auf blaue Leuchtstoff-Punkte treffen *(Abb. 1.7.2.5)*. Entsprechendes gilt für die roten und grünen Punkte. (Mit der Konvergenz-Einstellung läßt sich das genau justieren.) Die Helligkeitssteuerung erfolgt für jede Farbe getrennt über die zugehörige Katode. Ebenso ist für jede Farbe ein getrennter Video-Verstärker notwendig.

Die Erzeugung der für Bildröhren nötigen Hochspannung erfolgt gewöhnlich aus der Horizontal-Ablenkfrequenz (15625 Hz bei der europäischen Fernsehnorm). Diese hohe Frequenz bietet den Vorteil, daß der erforderliche Transformator räumlich klein ausfallen kann. Je nach Größe der Bildröhre muß der Horizontal-Ablenkverstärker ohnehin eine Leistung von einigen zehn Watt liefern, so daß die zusätzliche Leistung für die Hochspannung kaum ins Gewicht fällt: Der Anodenstrom der Bildröhre beträgt weniger als 1 mA.

Die Video-Eingänge von Bildschirm-Monitoren sind weitgehend standardisiert. Schwarzweiß-Monitore verfügen meist über einen Eingang mit 75 Ω Scheinwiderstand (Impedanz), das nötige Eingangssignal ist 1,5 Vss (Volt Spitze-Spitze, *Abb. 1.7.2.6*). Als Stecker dienen koaxiale Cinch- oder BNC-Ausführungen, als Verbindung zum Computer sollte ein 75-Ω-Kabel verwendet werden. Bei den üblichen

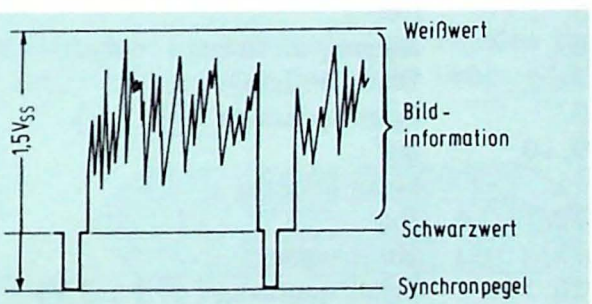


Abb. 1.7.2.6
So sieht ein Videosignal aus, wie es am Eingang eines Monitors vorhanden sein muß

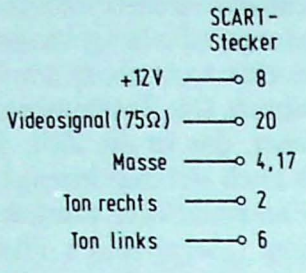


Abb. 1.7.2.7 Einspeisen eines FBAS- bzw. BAS-Signals über einen SCART-Stecker

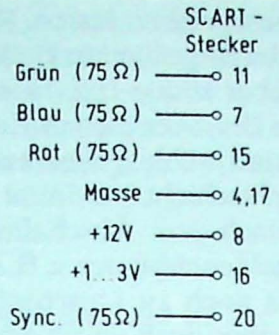


Abb. 1.7.2.8 Einspeisen eines RGB-Signals über den SCART-Stecker; die Qualität ist hierbei besser als beim FBAS-Signal

Kabellängen von weniger als 2 m ist allerdings auch der Einsatz von 50- oder 60-Ω-Koaxkabeln ohne sichtbare Auswirkungen auf die Wiedergabe.

Bei Farbmonitoren hat sich heute weitgehend der SCART-Stecker durchgesetzt. Er ist ursprünglich für die Verbindung von Farbfernsehgeräten mit Videorecordern entwickelt worden und besitzt daher auch zwei Tonleitungen (für Stereoton) sowie eine Schaltleitung, mit der sich das Fernsehgerät von Empfang via Antenne auf den SCART-Eingang umschalten läßt (Pin 8).

Dem SCART-Eingang kann entweder über Pin 20 ein PAL-FBAS- bzw. Schwarzweiß-Videosignal zugeführt werden *(Abb. 1.7.2.7)*, oder die drei Signale für Rot, Grün und Blau werden getrennt an die Pins 15, 11 und 7 gelegt, wobei Pin 20 nur noch als Eingang für das Synchronisations-Signal dient *(Abb. 1.7.2.8)*. Die getrennte RGB-Zuführung ist qualitativ wesentlich günstiger, da durch die Zusammenfassung zu einem einzigen FBAS-Signal die Bildschärfe abnimmt.

Die Bedeutung der 20 Pins des SCART-Steckers ist folgende:

Pin	Bedeutung	Pin	Bedeutung
1	Ausgang für rechtes Tonsignal		
2	Eingang für rechtes Tonsignal		
3	Ausgang für linkes Tonsignal		
4	Masse		
5	frei		
6	Eingang für linkes Tonsignal		
7	Blau-Eingang		
8	Schaltspannung 1 (+12 V)		
9, 10	frei		
11	Grün-Eingang	17	Masse
12, 13, 14	frei	18	frei
15	Rot-Eingang	19	Video-Ausgang
16	Schaltspannung 2 (+1...3 V)	20	Video-Eingang

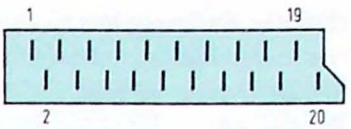


Abb. 1.7.2.9 Zählweise der Kontakte am SCART-Stecker

Die Anordnung der Pins und der Querschnitt des Steckers geht aus *Abb. 1.7.2.9* hervor. Es ist unbedingt ratsam, für die Ton-, Farb- und Video-Signale abgeschirmte Leitungen zu verwenden, um Brummen und Bildstörungen durch kapazitive Einstreuungen zu vermeiden.

1.7.3 Schnittstellen

Mit der Erwähnung des SCART-Steckers wären wir bereits bei einem weiteren wichtigen Thema, der Verbindung des Computers mit seinen Peripheriegeräten. Hierfür haben sich je nach Art der Daten und der nötigen Geschwindigkeit unterschiedliche Standards entwickelt, so daß es im Idealfall möglich ist, einen Computer der Firma X mit einem Drucker der Firma Y zu verbinden. Im Idealfall deshalb, weil die genannten Standards noch genug Spielraum dafür lassen, welches Steuerzeichen beim Drucker z.B. auf eine andere Schriftart oder auf Grafik umschalten soll. Diese kleinen Probleme sind jedoch gewöhnlich per Software zu lösen; davon wird später die Rede sein.

Die Centronics-Parallel-Schnittstelle

Der Druckerhersteller Centronics führte eine Schnittstelle ein, die von zahlreichen anderen Firmen übernommen wurde und heute einen Quasi-Standard für Drucker darstellt. Sie arbeitet mit TTL-Pegeln; deshalb sollte die Leitungslänge unter 2 m bleiben. Ferner sollten die einzelnen Signalleitungen mit Masseleitungen verdrillt werden (twisted pairs). Dies wird schon durch die Stecker-Pinbelegung unterstützt, da jeder signalführenden Leitung ein Massepin gegenüberliegt.

DATA 1 ist an dem 36poligen Amphenol-Stecker *(Abb. 1.7.3.1)*, AMP 57-30360) das niederwertigste Datenbit. Das höchstwertige Bit 8 wird von vielen Druckern ignoriert, da der ASCII-Zeichensatz ja mit sieben Bits auskommt. Verwendet der Drucker das achte Bit für Sonderfunktionen, wird es aber vom Computer nicht erzeugt, so muß dieser Pin mit Masse (log. 0) verbunden werden.

Es ist für einen einwandfreien Betrieb nicht erforderlich, alle Schnittstellen-Signale zu erzeugen bzw. auszuwerten. Im Prinzip genügen bereits die sieben oder

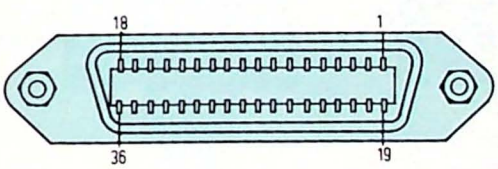


Abb. 1.7.3.1 Der Centronics-Stecker stellt eine parallele Drucker-Schnittstelle dar

Tabelle: Belegung des Centronics-Steckers

Signal-pin	Masse-pin	Signal	Richtung	Bedeutung
1	19	$\overline{\text{STROBE}}$	Ein	$\overline{\text{STROBE}}$ wird gepulst, um Daten an den Drucker zu übergeben. Impulsbreite > 0,5 µs
2	20	DATA 1	Ein	
3	21	DATA 2	Ein	
4	22	DATA 3	Ein	Diese Signale repräsentieren die 8 Datenbits. Sie werden übernommen, wenn $\overline{\text{STROBE}}$ Low ist. (High = logisch 1, Low = logisch 0)
5	23	DATA 4	Ein	
6	24	DATA 5	Ein	
7	25	DATA 6	Ein	
8	26	DATA 7	Ein	
9	27	$\overline{\text{DATA 8}}$	Ein	
10	28	$\overline{\text{ACKNLG}}$	Aus	Impuls von ca. 12 µs Länge. Low besagt, daß die Daten verarbeitet wurden und der Drucker empfangsbereits ist.
11	29	BUSY	Aus	Solange BUSY high ist, ist der Drucker nicht empfangsbereit. Busy ist high, wenn: 1. Daten empfangen werden 2. Während des Druckens 3. Im OFF-LINE-Modus 4. Im Fehlerfall
12	30	PE	Aus	High bedeutet „Papier zu Ende“
13	—	—	—	High-Level (+5 V über 3,3 kΩ)
14	—	$\overline{\text{AUTO FEED XT}}$	Ein	Low = automatisches LF High = kein automatisches LF (siehe 7.2.2)
15	—	NC	—	unbenutzt
16	—	0V	—	Massepegel (Logik)
17	—	CHASSIS GND	—	Masse Drucker. Druckermasse und Logikmasse sind voneinander isoliert.
18	—	NC	—	unbenutzt
19—30	—	$\overline{\text{GND}}$	—	Massepins für 1—12
31	—	$\overline{\text{INIT}}$	Ein	Wenn dieser Pin mindestens 50 µs auf Low geht, wird der Drucker normiert.
32	—	$\overline{\text{ERROR}}$	Aus	Dieser Pin geht auf Low, wenn der Drucker 1. offline ist, 2. das Papier zu Ende ist, 3. einen Fehler entdeckt.
33	—	GND	—	wie 19—30
34	—	NC	—	unbenutzt
45	—	—	—	wie 13
36	—	$\overline{\text{SLCT IN}}$	Ein	Selektion des Druckers. Codes DC 1/ DC 3 funktionieren nur, wenn dieser Pin High ist

acht Datenbits sowie die Signale BUSY und STROBE. Alle anderen Pins können unbeschaltet bleiben; sie sind teilweise leider auch von Hersteller zu Hersteller unterschiedlich verwendet.

Mit einer fallenden Impulsflanke an STROBE werden die anstehenden Daten vom Drucker übernommen. Während des Druckvorgangs meldet er dann über ein High-Signal an BUSY, daß er gerade intern beschäftigt ist und keine weiteren Daten aufnehmen kann.

V.24 und RS-232C

Die CCITT-Norm V.24 legt die Eigenschaften einer seriellen Schnittstelle für asynchrone Start-Stop-Übertragung fest. Sie entspricht der amerikanischen RS-232C-Norm.

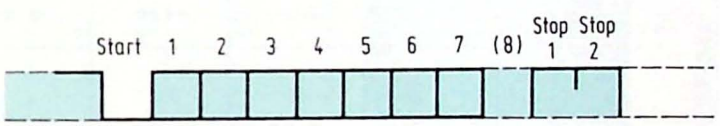
Die Zeichen werden bitweise über eine einzige Datenleitung übertragen *(Abb. 1.7.3.2)*. Dabei sind folgende Geschwindigkeiten üblich:

110 Bit/s	2400 Bit/s
150 Bit/s	4800 Bit/s
300 Bit/s	9600 Bit/s
600 Bit/s	19200 Bit/s
1200 Bit/s	

Die Schnittstelle erlaubt den Datenverkehr in beiden Richtungen. Es sind deshalb alle Leitungen zweimal vorhanden, für jede Richtung einmal. Als Stecker wird eine 25polige Miniatur-D-Verbindung verwendet (z.B. Cannon 7529, *Abb. 1.7.3.3*). Die wichtigsten Leitungen sind:

E2	ist die gemeinsame Masseleitung für alle Signale.
D1	führt die Daten vom Computer zum Peripherie- bzw. Datenendgerät.
D2	führt die Daten vom Peripherie- bzw. Datenendgerät zum Computer.
S2	gibt dem Peripheriegerät bekannt, daß der Computer bereit ist.
M2	gibt dem Computer bekannt, daß das Peripheriegerät bereit ist.

Abb. 1.7.3.2
Serielle Datenübertragung an einer V.24- oder TTY-Schnittstelle



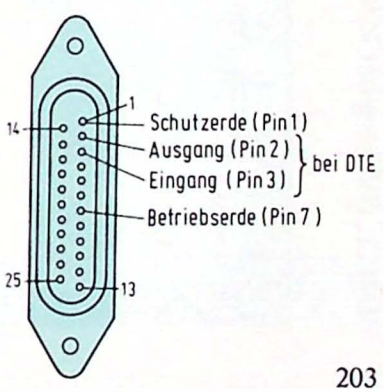


Abb. 1.7.3.3
Die wichtigsten Pins eines V.24-Steckers

Tabelle: Belegung des V.24-Steckers

	Kurzzeichen			Stecker-belegung	Beschreibung		Richtung	
	CCITT V.24	EIA RS 232	DIN 66020		Deutsch	Englisch	Modem (DCE)	Terminal (DTE)
Erde	101	AA	E 1	1	Schutzerde	Protective ground		
	102	AB	E 2	7	Signalerde/Betriebserde	Signal ground/Common return		
Daten	103	BA	D 1	2	Sendedaten	Transmitted data (TD)	←	
	104	BB	D 2	3	Empfangsdaten	Received data (RD)		→
Steuer- und Meldesignale	105	CA	S 2	4	Sendeteil einschalten	Request to send (RTS)	←	
	106	CB	M 2	5	Sendebereitschaft	Clear to send (CTS)		→
	107	CC	M 1	6	Betriebsbereitschaft	Data set ready (DSR)		→
	108.1		S 1.1	20	Übertragungsleitung anschalten	Connect data set to line	←	
	108.2	CD	S 1.2	20	Terminal betriebsbereit	Data terminal ready (DTR)	←	
	125	CE	M 3	22	Ankommender Ruf	Ring indicator (RI)		→
	109	CF	M 5	8	Empfangssignalpegel	Received line signal detector (DCD) carrier detector		→

	110	CG	M 6	21	Empfangsgüte	Signal quality detector
	111	CH	S 4	23	Übertragungsgeschwindigkeit (Wahl vom Terminal)	Data signal rate selector (DTE)
	112	CI	M 4	23	Übertragungsgeschwindigkeit (Wahl vom Modem)	Data signal rate selector (DCE)
	126	CK	S 5	11	Wahl Sendefrequenz (200-Baud-Modem)	Select transmit frequency (200 baud modem)
Takte	113	DA	T 1	24	Sendeschrittakt von DEE	Transmitter signal element timing (Transmit clock to modem DTE)
	114	DB	T 2	15	Sendeschrittakt von DÜE	Transmitter signal element timing (TC) (Transmit clock from modem DCE)
	115	DD	T 4	17	Empfangsschrittakt	Receiver signal element timing (RC) Receive clock
Zusatzkanal	118	SBA	HD 1	14	Sendedaten Rückkanal	Secondary transmitted data
	119	SBB	HD 2	16	Empfangsdaten Rückkanal	Secondary received data
	120	SCA	HS 2	19	Rückkanal Sendeteil einschalten	Secondary request to send
	121	SCB	HM 2	13	Rückkanal Sendebereitschaft	Secondary clear to send
	122	SCF	HM 5	12	Rückkanal Empfangssignalpegel	Secondary Carrier detector
Frei				9/10 11/18/ 25	Zur Verwendung für Prüfgeräte Nicht belegt	Reserved for data set testing Unassigned

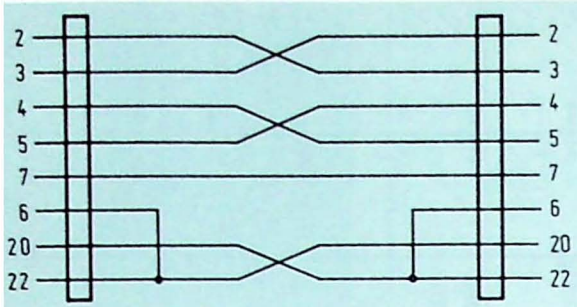


Abb. 1.7.3.4
So sieht ein „Nullmodem" aus

Die V.24-Schnittstelle arbeitet nicht mit TTL-Pegeln, sondern normalerweise mit −12 V für logisch 1 und +12 V für log. 0 (!). Maximal ist der Bereich −15...+15 V erlaubt. Der Pegelbereich −3...+3 V ist „verboten", d.h. der Absolutwert der Spannung muß an einem V.24-Eingang auf jeden Fall höher als 3 V sein. Wegen des relativ hohen Spannungshubes funktioniert die Übertragung auch ohne abgeschirmte Leitungen sehr störsicher.

Die serielle Zeichenübertragung auf den Datenleitungen stellt den eigentlichen Datenbits ein Startbit voraus, das log. 0 ist (+12 V). Weil der Ruhepegel der Leitung log. 1 = −12 V ist, kann so der Zeichenbeginn einwandfrei erkannt werden. Um sicherzustellen, daß das Startbit des folgenden Zeichens auch dann erkannt wird, wenn das letzte Datenbit des gerade gesendeten Zeichens ebenfalls Null ist, trennt man zwei Zeichen durch ein oder zwei Stopbits mit der Ruhelage-Polarität 1. Bei den Steuerleitungen gilt +12 V = ein.

Von den Datenbits wird das niederwertigste stets zuerst gesendet. Für die Übertragung von ASCII-Zeichen genügen eigentlich sieben Datenbits; das achte (höchstwertige, zuletzt gesendete) setzt man dann entweder fest auf 0 bzw. 1 oder verwendet es als Prüfbit: Um Übertragungsfehler erkennen zu können, bildet man die Summe aller Bits. Ist sie ungerade, setzt man das Prüfbit auf 1, andernfalls auf Null (ungerade Parität). Manchmal wird auch mit gerader Parität gearbeitet; das Prüfbit ist dann invertiert.

Die Fehlererkennung durch ein Paritätsbit funktioniert allerdings nur dann, wenn die Zahl der Fehler in einem Zeichen ungerade ist (z.B. 1). Treten beispielsweise zwei Bitfehler auf, so werden sie durch Paritätsprüfung nicht erkannt. Bei Übertragungen, wo es auf absolute Fehlerfreiheit ankommt, ist es deshalb sinnvoll, sicherere Verfahren zur Fehlerprüfung (z.B. Prüfsummen oder CRC = Cyclic Redundancy Check) oder zur automatischen Fehlerkorrektur (z.B. Hamming-Codierung oder Kreuzparität) zu verwenden.

Die Belegung der V.24-Stecker ist für Computer (sog. Datenendeinrichtungen) und Peripheriegeräte (z.B. Modems, Drucker usw.) absichtlich unterschiedlich definiert, damit sie unmittelbar miteinander verbunden werden können. Denn an dem Stift, an den der Computer Zeichen sendet, muß sie das Peripheriegerät, z.B. ein Drucker, ja empfangen. Schwierigkeiten können daher dann auftreten, wenn man zwei Computer miteinander verbinden will: Dann ist ein sogenanntes „Null-

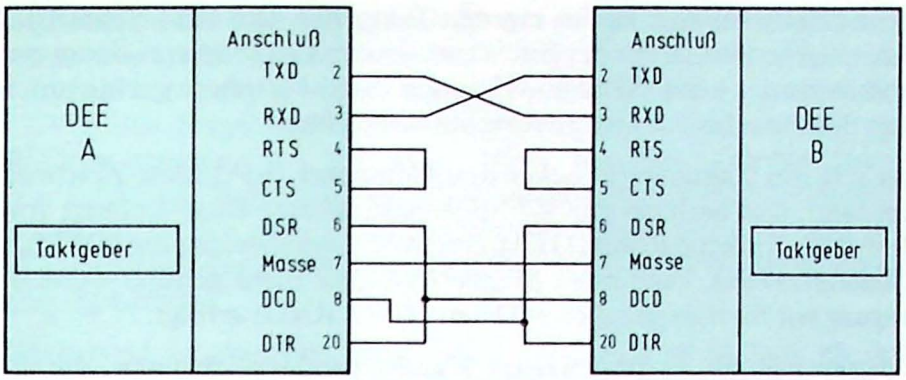


Abb. 1.7.3.5 Verbinden zweier Computer oder Terminals über ihre V.24-Schnittstellen bei Verwendung ihres eigenen Taktgebers

modem" nötig *(Abb. 1.7.3.4)*. Das ist ein Adapterstück oder ein Kabel mit zwei Steckern, das die entsprechenden Steckerpins miteinander vertauscht.

Gewisse Unklarheiten über die genaue Steckerbelegung können auftreten, wenn nicht ganz klar ist, ob ein Gerät eher als Datenendeinrichtung (DEE) oder als Datenübertragungs-Einrichtung (DÜE oder DCE = Data Communication Equipment) betrachtet werden soll. Dann hilft nur ein Blick in die jeweilige Geräte-Dokumentation.

In den meisten Fällen erzeugt jedes Gerät seinen für den Datentransfer nötigen Takt intern selbst. Die prinzipielle Verbindung zweier Datenendeinrichtungen (DEE) geht aus *Abb. 1.7.3.5* hervor. Einige Leitungen müssen dabei, wie erwähnt, gekreuzt werden.

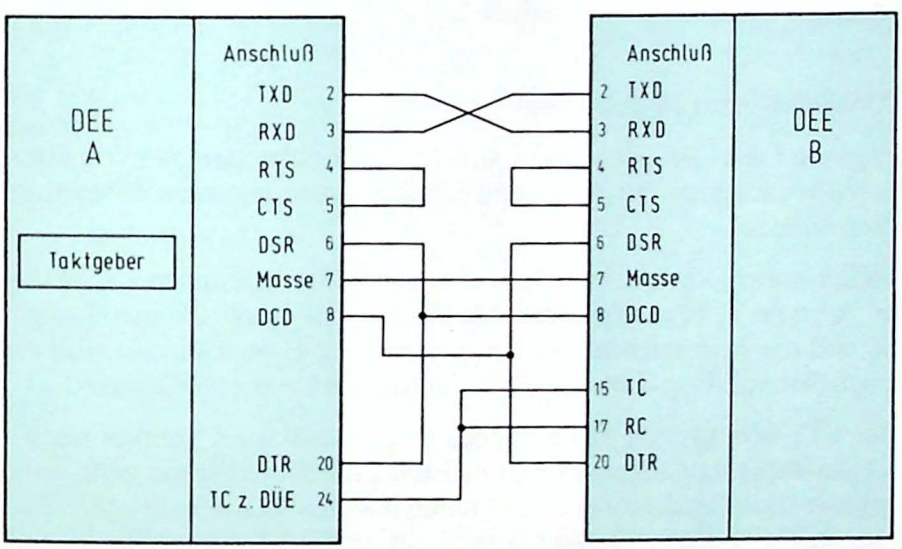


Abb. 1.7.3.6 Verbinden zweier Computer oder Terminals, wenn nur das links gezeichnete Gerät einen Taktgeber besitzt

Manche Geräte besitzen keinen eigenen Taktgeber. Die V.24-Schnittstelle definiert daher eigene Pins, damit der Bittakt von einem Gerät dem anderen zur Verfügung gestellt werden kann. Nicht alle Computer und Peripheriegeräte unterstützen allerdings diese in *Abb. 1.7.3.6* verdeutlichte Möglichkeit.

Damit z.B. ein langsamer Drucker den Computer bei dessen Zeichenausgabe bremsen kann, sind auch bei der V.24-Schnittstelle eine Busy-Leitung vorhanden (Pin 20 = Data Terminal Ready, DTR). Bei Datenfernübertragung (DFÜ), z.B. mit einem Telefonmodem, kann diese Möglichkeit aber nicht genutzt werden, da die Übertragung nur für die eigentlichen Daten (TXD, RXD) erfolgt.

Für diesen Fall gibt es zwei weitere Handshake-Möglichkeiten, die allerdings nicht von der Hardware, sondern in der Software (vom DFÜ-Steuerprogramm) abgewickelt werden.

Die gebräuchlichste davon ist das XON-XOFF-Protokoll. Dabei werden die beiden ASCII-Steuerzeichen DC1 (XON = hex 11 = CTRL-Q) und DC3 (XOFF = hex 13 = CTRL-S) verwendet. Mit XON meldet der Datenempfänger, daß er zur Aufnahme weiterer Zeichen bereit ist; mit XOFF stoppt er die Sendung seiner Gegenstation, während er intern z.B. mit dem Ausdrucken oder dem Abspeichern beschäftigt ist.

Die ETX-ACK-Prozedur funktioniert etwas anders (ETX = hex 03 = End of Text = Textende; ACK = hex 06 = Acknowledge = Bestätigung). Der Computer sendet hier einen ganzen Datenblock, der mit ETX abgeschlossen wird. Sobald er von der Gegenstation komplett verarbeitet (z.B. ausgedruckt) ist, sendet diese das Steuerzeichen ACK zurück, woran der Computer erkennt, daß er nun den nächsten Zeichenblock (z.B. eine Zeile) senden darf. Die ETX-ACK-Prozedur setzt empfangsseitig einen Pufferspeicher voraus, da der Empfänger ja die Übertragung während eines Datenblocks nicht anhalten kann.

Die TTY-Schnittstelle (20-mA-Loop)

Für sehr große Leitungslängen eignet sich die V.24-Schnittstelle nicht allzu gut, da induktiv (elektromagnetisch) eingekoppelte Störspannungen zu Übertragungsfehlern führen können.

Wesentlich unkritischer ist in dieser Beziehung die sogenannte TTY-Schnittstelle (TTY = Teletype = Fernschreiber). Man nennt sie auch 20-mA-Loop (Stromschleife), und das kennzeichnet sie schon recht gut: Eine logische Null entspricht hier keinem Stromfluß, und bei log. 1 fließt ein Strom von etwa 20 mA.

Bei der TTY-Schnittstelle gibt es weder einen genormten Stecker noch standardisierte Handshake-Leitungen. In den meisten Fällen wird sogar ganz ohne Handshake-Signale (Busy o.ä.) gearbeitet, es handelt sich also um eine echte Zweidraht-Übertragung. Das Format ist ebenso wie bei der V.24-Schnittstelle asynchron-seriell mit Start- und Stopbits. Die Ruhelage ist auch hier log. 1 entsprechend 20 mA Stromfluß.

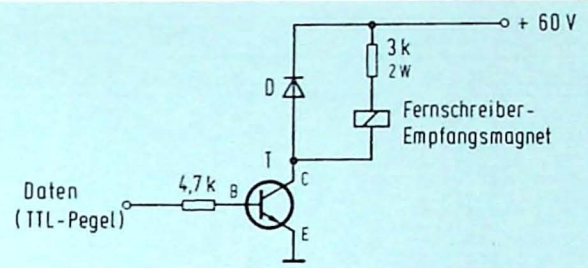
Im Gegensatz zu V.24 gibt es bei der 20-mA-Stromschleife kein definiertes Bezugspotential, also keine gemeinsame Masse oder Erde. Dies ergibt sich von selbst aus der früher bei mechanischen Fernschreibern üblichen Ansteuer-Schaltung *(Abb. 1.7.3.7)*: Der Empfangsmagnet des Fernschreibers liegt zwischen der recht hohen Betriebsspannung von z.B. 60 ...100 V und dem Collector eines Schalttransistors, an dessen Basis das Datensignal anliegt. Ein Collector-Vorwiderstand begrenzt den Strom auf etwa 20 mA, und eine Diode verhindert eine Beschädigung des Transistors durch die in der Spule des Empfangsmagneten entstehende Rückschlag-Spannung.

Der Witz dabei ist, daß der Strom von 20 mA im wesentlichen nur durch die Höhe der Betriebsspannung und den Wert des Vorwiderstands bestimmt wird, da alle anderen Faktoren wie Innenwiderstand des Empfangsmagneten oder Leitungswiderstand selbst bei sehr großer Leitungslänge eine vernachlässigbare Rolle spielen. Da die Betriebsspannung recht hoch ist, können auch eingekoppelte Störspannungen bis zu einigen Volt keine Übertragungsfehler verursachen.

Weil es bei zwei durch eine 20-mA-Stromschleife verbundenen Geräten kein gemeinsam gültiges Bezugspotential gibt, werden meist Optokoppler eingesetzt, so daß Sender und Empfänger galvanisch völlig voneinander getrennt sind *(Abb. 1.7.3.8)*. Das sind Bauelemente, die in einem einzigen Gehäuse eine Leuchtdiode (LED) und einen Fototransistor vereinen. Wenn man Strom durch die LED fließen läßt, beleuchtet diese den Fototransistor, so daß dessen Collector-Emitter-Strecke leitend wird.

Da die TTY-Schnittstelle keine Handshake-Leitungen und keine Steckerbelegung normt, hat sie sich im Vergleich zum V.24-Standard nur für Spezialanwendungen durchsetzen können.

Abb. 1.7.3.7
Ansteuerung des Empfangsmagneten eines Fernschreibers



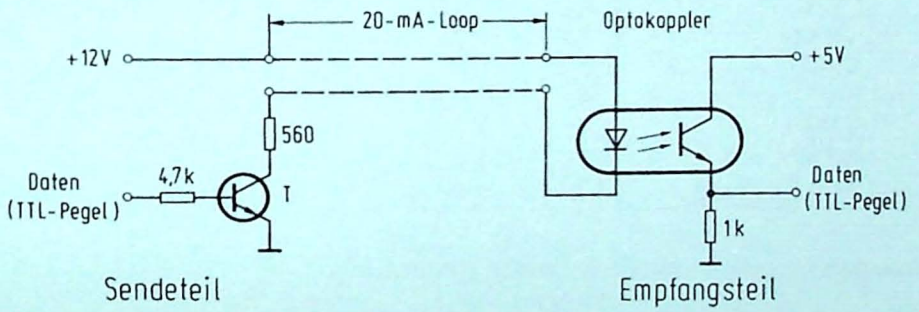


Abb. 1.7.3.8 Galvanische Trennung von TTY-Schnittstellen mit einem Optokoppler

Der IEC-Bus (IEEE-488)

Die bisher aufgeführten Schnittstellen waren ausschließlich dafür bestimmt, zwei (und nicht mehr) Geräte miteinander zu verbinden. Den Ausdruck „Bus" kennen wir schon vom Adressen- und Datenbus innerhalb eines Mikrocomputers; er bedeutet, daß mehrere Peripheriebausteine gleichberechtigt über eine gemeinsame Gruppe von Leitungen mit der Zentraleinheit verbunden werden. In der Tat stellt auch der IEC-Bus eine Gruppe von Leitungen dar, an die man bis zu 15 Peripheriegeräte anschließen kann. Der Computer kann jedes Gerät einzeln adressieren. Nur wenn ein Gerät seine Adresse erkannt hat, kümmert es sich um die nachfolgenden Daten.

Der Bus, um den es hier geht, ist unter unterschiedlichen Namen bekannt, nämlich als IEC-Bus und IEEE-488-Bus. Beide Normen sind elektrisch gleichwertig, verwenden aber unterschiedliche Stecker mit 25 bzw. 24 Polen. Die Signale werden mit negativer Logik und TTL-Pegel angelegt (log. 0 = 5 V, log. 1 = 0 V). Die Gesamt-Kabellänge ist deshalb auf 20 m begrenzt.

Der IEC-Bus arbeitet wie die Centronics-Schnittstelle parallel, d. h. die acht Bits eines Bytes bzw. Zeichens werden gleichzeitig an acht Datenleitungen gelegt. Diese Datenleitungen sind hier allerdings bidirektional, da eine Übertragung in beiden Richtungen möglich ist (vom Computer zum Peripheriegerät und umgekehrt).

Tabelle: IEC-Bus-Steckerbelegung

IEC 66.22		IEEE	IEC 66.22		IEEE
Kontakt	Signal	Kontakt	Kontakt	Signal	Kontakt
1	DIO 1	1	14	DIO 5	13
2	DIO 2	2	15	DIO 6	14
3	DIO 3	3	16	DIO 7	15
4	DIO 4	4	17	DIO 8	16
5	REN	17	18	Masse/GND	—
6	EOI	5	19	Masse (6)	18
7	DAV	6	20	Masse (7)	19
8	NRFD	7	21	Masse (8)	20
9	NDAC	8	22	Masse (9)	21
10	IFC	9	23	Masse /GND	—
11	SRQ	10	24	Masse (11)	23
12	ATN	11	25	Masse (12)	—
13	Abschirmung	12	—	Masse (10)	22

Die einzelnen Leitungen haben folgende Bedeutung:

DIO 1...7 (Data In/Out): Bidirektionale Datenleitungen, log. 1 = 5 V.

ATN (Attention): Der Controller (normalerweise der Computer) legt diese Leitung auf Low, solange er Daten sendet. Wenn ATN auf High liegt, kann ein vorher adressiertes Gerät Daten zum Controller senden.

DAV (Data Valid): Die an DIO 1...8 liegenden Daten sind gültig, solange DAV auf Low liegt.

EOI (End of Information): Der Datensender (Talker; Computer oder Peripheriegerät) legt EOI auf Low, sobald das letzte Zeichen übertragen ist.

NDAC (No Data Accepted): Der „Listener" (Datenempfänger; kann der Computer oder ein Peripheriegerät sein) legt NDAC auf Low, solange er Daten aufnimmt, und auf High, sobald er ein Byte übernommen hat.

NRFD (Not Ready for Data): Low zeigt dem Controller an, daß ein oder mehrere Geräte nicht bereit sind, Daten zu empfangen.

SRQ (Service Request): Ein Peripheriegerät, das Daten für den Controller bereit hält, zieht diese Leitung auf Low. SRQ geht wieder auf High, sobald ein Datentransfer zustande kommt.

REN (Remote Enable): Normalerweise im Controller mit Masse verbunden. Bei einem System mit mehreren Controllern legt nur der jeweils aktive REN auf Low-Pegel.

Der Handshake-Ablauf der wichtigsten IEC-Bus-Signale wird nochmals in *Abb. 1.7.3.9* verdeutlicht. Er läßt sich im Prinzip mit einem geeigneten Steuerprogramm rein softwaremäßig realisieren, allerdings ist die erreichbare Transfer-Geschwin-

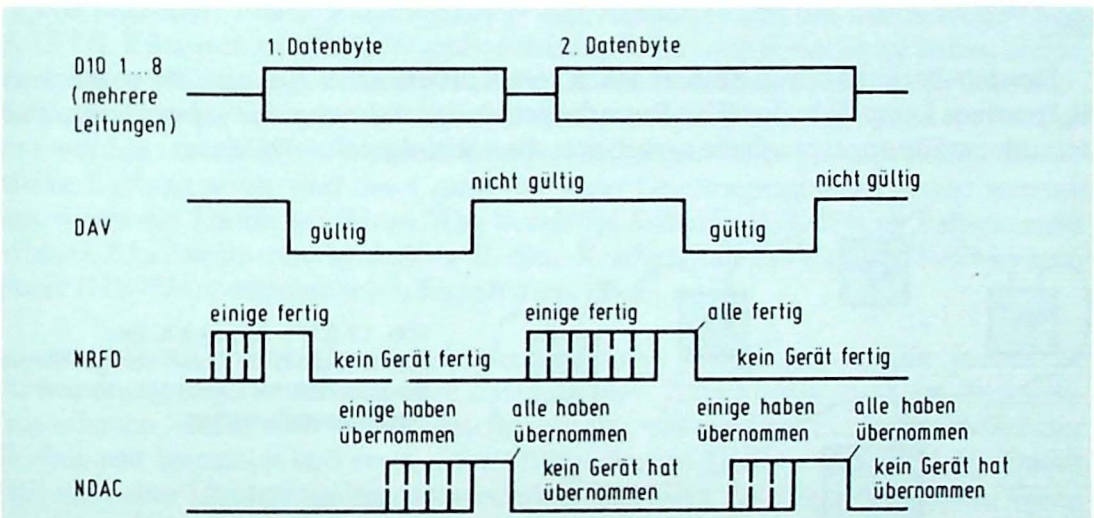


Abb. 1.7.3.9 Zeitablauf der Übertragung zweier Bytes über den IEC-Bus

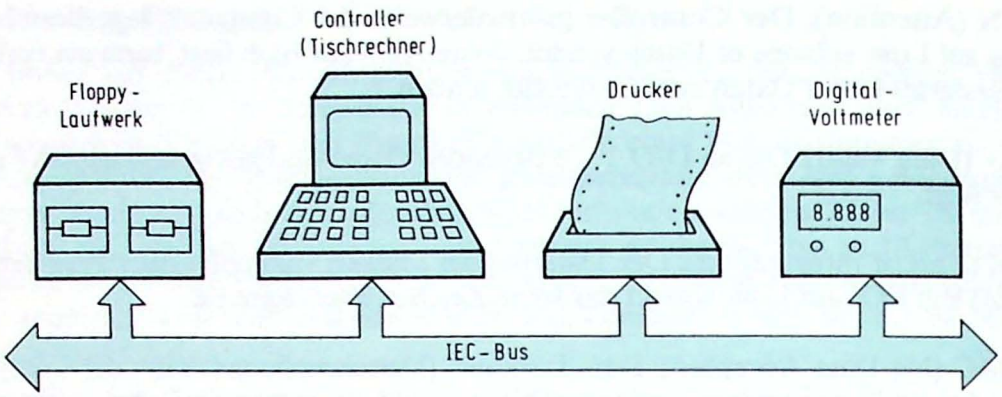


Abb. 1.7.3.10 Typische Gerätekonfiguration in einem IEC-Bus-System

digkeit dann begrenzt. Für den bidirektionalen Datenaustausch sind spezielle IEC-Bus-Treiber-Bausteine erhältlich; normale I/O-Ports würden nicht ausreichen, um 15 Peripheriegeräte zu treiben.

Der IEC-Bus hat sich insbesondere bei meßtechnischen Anwendungen verbreiten können *(Abb. 1.7.3.10)*. So ist es z.B. problemlos möglich, ein oder mehrere Meßgeräte, ein Floppy-Laufwerk zum Speichern der Meßwerte und einen Drukker zum Protokollieren der Ergebnisse über einen einzigen, gemeinsamen Bus mit dem Computer als Bus-Controller zu verbinden. Der Vorteil liegt auf der Hand: Statt vieler Schnittstellen (für jedes Peripheriegerät eine) benötigt der Computer nur noch einen einzigen Bus-Anschluß.

Die Interface-Loop (IL-Bus)

Der IEC-Bus benötigt relativ viele Leitungen, da er bitparallel arbeitet. Ferner ist er wegen der Verwendung von TTL-Pegeln nur für Leitungslängen bis 20 m geeignet.

Hewlett-Packard führte deshalb einen seriell arbeitenden Bus ein, die sogenannte Interface Loop. Wie der IEC-Bus arbeitet sie mit Adressen für jedes Gerät, und nur das gerade angesprochene verarbeitet die nachfolgenden Daten.

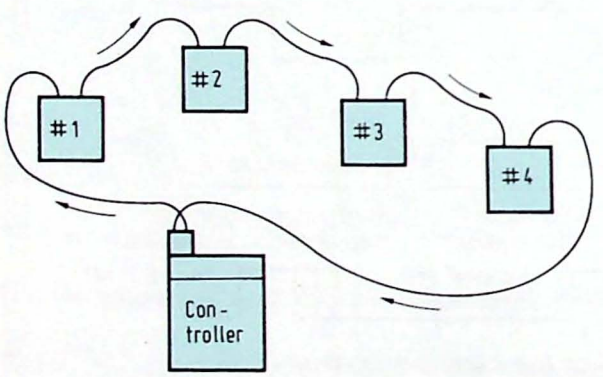


Abb. 1.7.3.11 Auf dem IL-Bus durchwandern die Daten den gesamten Ring, so daß der Controller jederzeit Fehler feststellen kann

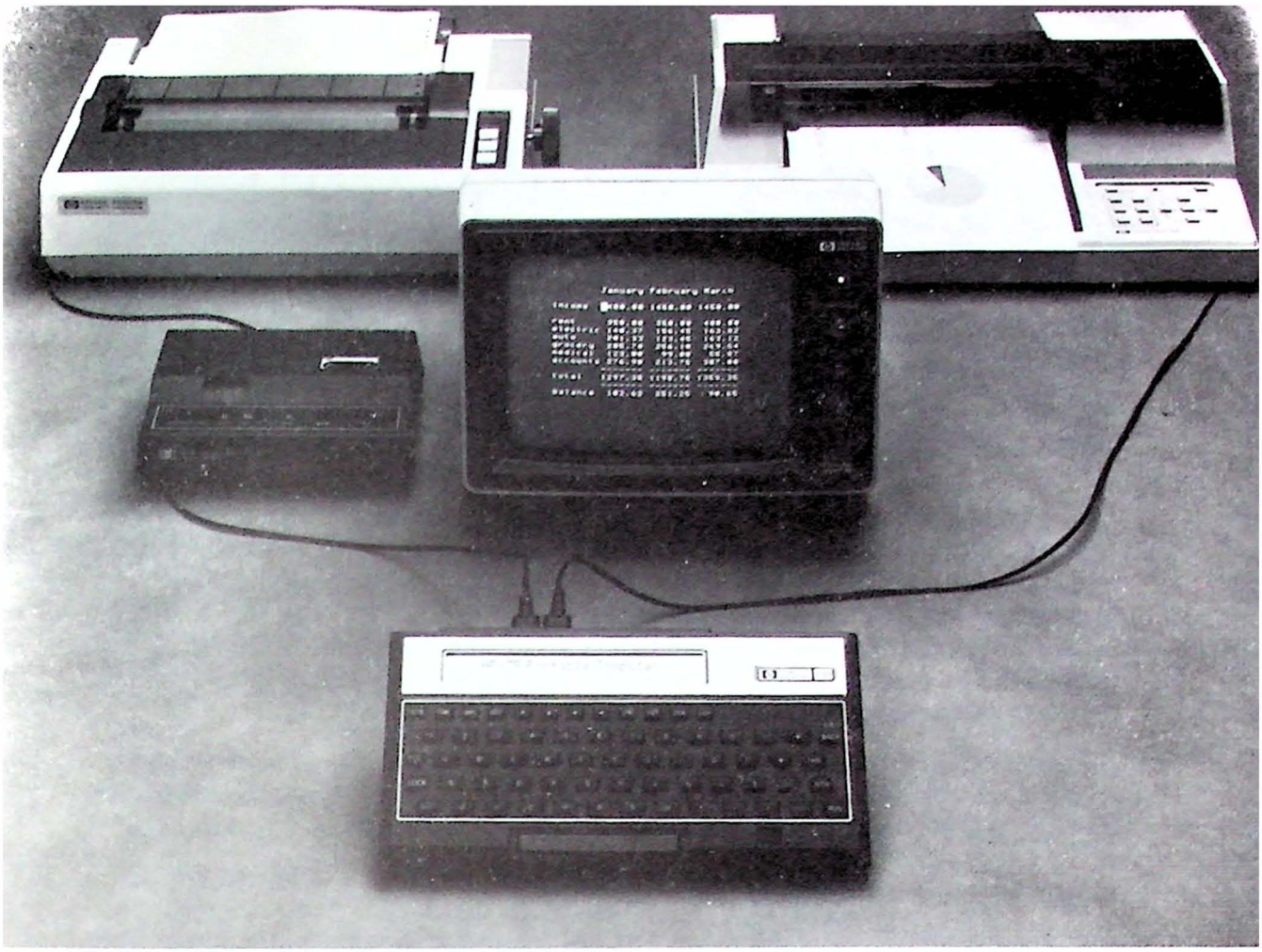


Abb. 1.7.3.12 Der Taschencomputer HP-75C wird über den IL-Bus mit Peripheriegeräten verbunden

Wie schon der Name Interface Loop sagt, handelt es sich um eine Schleife *(Abb. 1.7.3.11)*. Eine potentialfreie Zweidrahtleitung führt nacheinander zu jedem Gerät, und alle nicht angesprochenen geben die Daten unverändert weiter. Das adressierte Gerät hingegen verarbeitet zunächst die Daten und gibt sie erst dann weiter. Damit werden zwei Fliegen mit einer Klappe geschlagen: Es ist keine getrennte Handshake-Leitung nötig, und der Controller kann Übertragungsfehler sofort feststellen, wenn die Daten auf ihrem Weg durch die Schleife verändert zurückkommen. *Abb. 1.7.3.12* zeigt eine komplette IL-Bus-Konfiguration mit einem Taschencomputer (HP-75) sowie mehreren Peripheriegeräten.

Die Potentialfreiheit (keine gemeinsame Masse nötig) wird erreicht, indem die Ankopplung in jedem Gerät über einen kleinen Transformator erfolgt. Bekanntlich können Trafos aber keine Gleichspannung weitergeben. Deshalb arbeitet der IL-Bus mit Impulsen von etwa einer Mikrosekunde Dauer *(Abb. 1.7.3.13)*. Damit läßt sich eine Übertragungsgeschwindigkeit von rund 20 KByte/s erzielen. Wenn man eine Leitungsimpedanz von etwa 100 Ω einhält, kann man Entfernungen bis 100 m überbrücken.

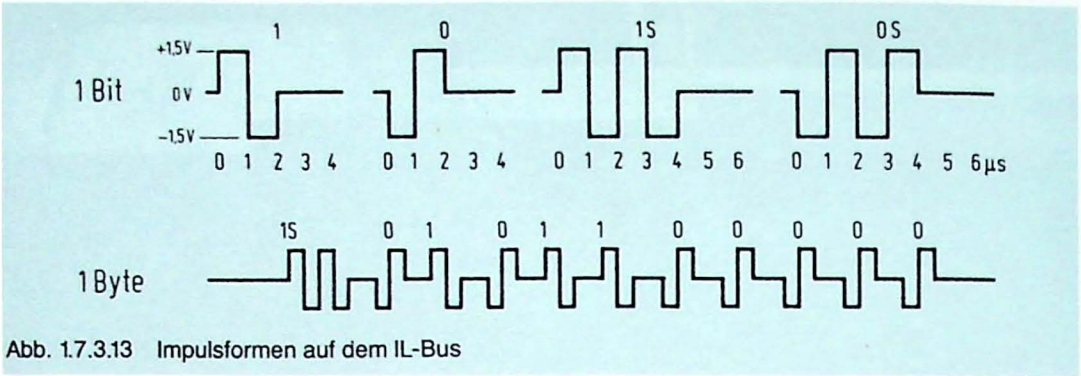


Abb. 1.7.3.13 Impulsformen auf dem IL-Bus

Das Verfahren verwendet fünf Adressenbits, so daß 31 Geräte adressiert werden können. Jedem übertragenen 8-Bit-Zeichen werden drei weitere für die Unterscheidung Adresse/Daten/Steuerbefehl vorausgestellt. Das erste Bit dient auch zur Synchronisation, das zweite gibt das Datenblock-Ende an und das dritte ermöglicht einem Peripheriegerät, dem Controller zu signalisieren, daß es Daten für ihn bereithält. Der Controller und alle Peripheriegeräte können Talker oder Listener sein, also Daten senden oder empfangen.

Wegen des geringen Strombedarfs ist der IL-Bus insbesondere für batteriebetriebene Geräte interessant, wo eine hohe Übertragungsgeschwindigkeit nicht im Vordergrund steht.

Floppy-Laufwerk-Anschlüsse

Von der Floppy-Disk-Aufzeichnung wird im folgenden Kapitel noch ausführlich die Rede sein. Hier soll es daher nur um die Schnittstellen zwischen handelsüblichen Laufwerken (ohne eingebauten Controller) und Computern gehen.

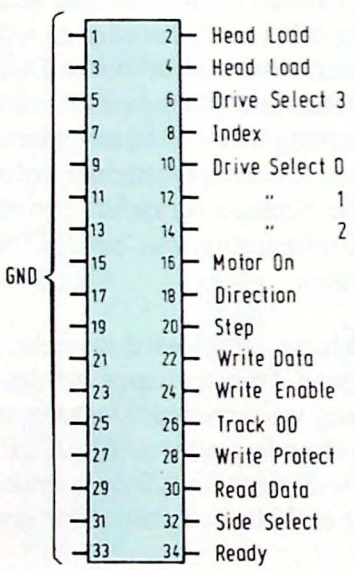


Abb. 1.7.3.14
Standard-Schnittstelle für 5,25-Zoll- und 3,5-Zoll-Floppy-Laufwerke

Dafür haben sich im wesentlichen zwei Standards herauskristallisiert. Laufwerke für 3,5- und 5,25-Zoll-Floppies besitzen gewöhnlich einen 34poligen Stecker *(Abb. 1.7.3.14)*, während 8-Zoll-Laufwerke mit einem 50poligen Stecker ausgestattet sind *(Abb. 1.7.3.15)*. Die Signale arbeiten mit TTL-Pegeln und negativer Logik (5 V = log. 0, 0 V = 1):

N.C.	Not connected (nicht angeschlossen, nicht verwendet)
Head Load	Bei Low wird der Schreib-Lese-Kopf abgesenkt
Drive Select	Auswahlsignal für eines von mehreren Laufwerken
Motor on	Laufwerksmotor einschalten
Direction	Richtung für Step
Step	Eine Spur weitergehen (Richtung: Direction)
Write Data	Schreib-Daten
Write Enable	Schreiben zulassen
Track 0	Spur 0 erreicht
Write Protect	Diskette ist schreibgeschützt
Read Data	Gelesene Daten (mit Taktsignal vermischt)
Side Select	Seite (bei doppelseitigen Laufwerken)
Ready	Laufwerk bereit
Index	Impuls vom Indexloch
Tracks>43	Spurnummer größer als 43
Dual Side Ind.	Doppelseitiges Laufwerk
Write Gate	Schreibfreigabe-Gatter

Ein Teil der Signale wurde auch bereits im Kapitel über Peripheriebausteine (Floppy-Controller) beschrieben.

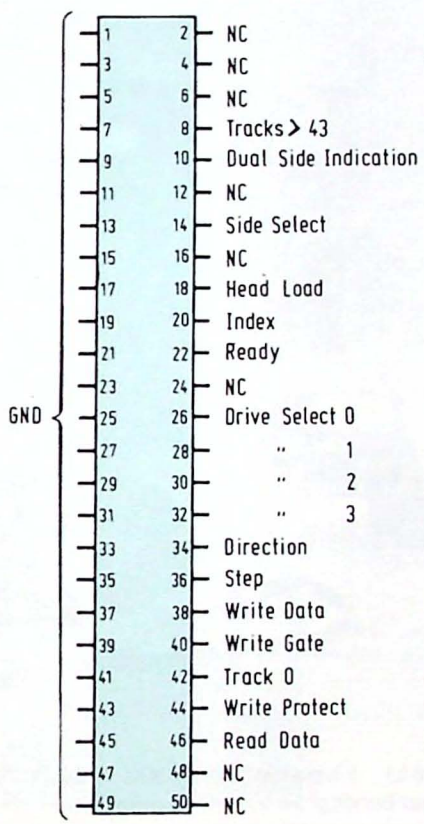


Abb. 1.7.3.15
Standard-Schnittstelle für 8-Zoll-Floppy-Laufwerke

1.8 Externe Speicherverfahren

Die vom Benutzer in den Arbeitsspeicher des Computers geschriebenen Daten gehen beim Abschalten der Betriebsspannung unwiederbringlich verloren. Ein batteriegepuffertes CMOS-RAM wäre bei der nötigen Größe andererseits zu teuer, um alle jemals erfaßten Programme und Daten zu speichern. Man verwendet daher zur Auslagerung von Daten und Programmen nichtflüchtige Verfahren, d.h. beim Abschalten der Betriebsspannung tritt kein Datenverlust auf. Das hierfür am häufigsten eingesetzte Prinzip ist die Floppy-Disk-Aufzeichnung.

1.8.1 Floppy-Disk-Aufzeichnung

Disketten (ein anderes Wort für Floppy Disks, deutsch „Schlappscheiben") sind runde, flexible Kunststoffscheiben, die mit einem magnetisierbaren Material be-

Abb. 1.8.1.1 Eisenoxid, Bindemittel und Lösungsmittel werden für die Herstellung magnetischer Datenträger benötigt

schichtet sind. Eisenoxid, Lösungsmittel und Bindemittel sind die wichtigsten Zutaten für die Dispersion, mit der Disketten (aber auch Computerbänder und Festplatten) beschichtet sind *(Abb. 1.8.1.1)*. Die zu speichernden Bits werden in elektromagnetische Impulse umgewandelt und somit die mikroskopisch kleinen Oxidteilchen magnetisiert. Teilweise wird statt Eisenoxid auch Chromdioxid eingesetzt.

Die runde Kunststoff-Folie ist weniger als ein Zehntel Millimeter dick, ihre magnetische Beschichtung sogar nur etwa 2,5 Tausendstel Millimeter *(Abb. 1.8.1.2)*, wesentlich weniger als die Dicke eines Menschenhaares, ja sogar weniger als der Durchmesser eines Staubkorns. Daraus ergibt sich schon, daß Disketten sehr empfindlich gegen Verschmutzung sind. Man umgibt sie deshalb mit einer schützenden Hülle, in der sie bei geringem Reibungswiderstand rotieren können *(Abb. 1.8.1.3)*. Die Hülle weist zur Abtastung mit einem Magnetkopf einen Schlitz auf, auf den man möglichst nicht fassen sollte. Die Hülle weist in der Mitte eine Bohrung auf,

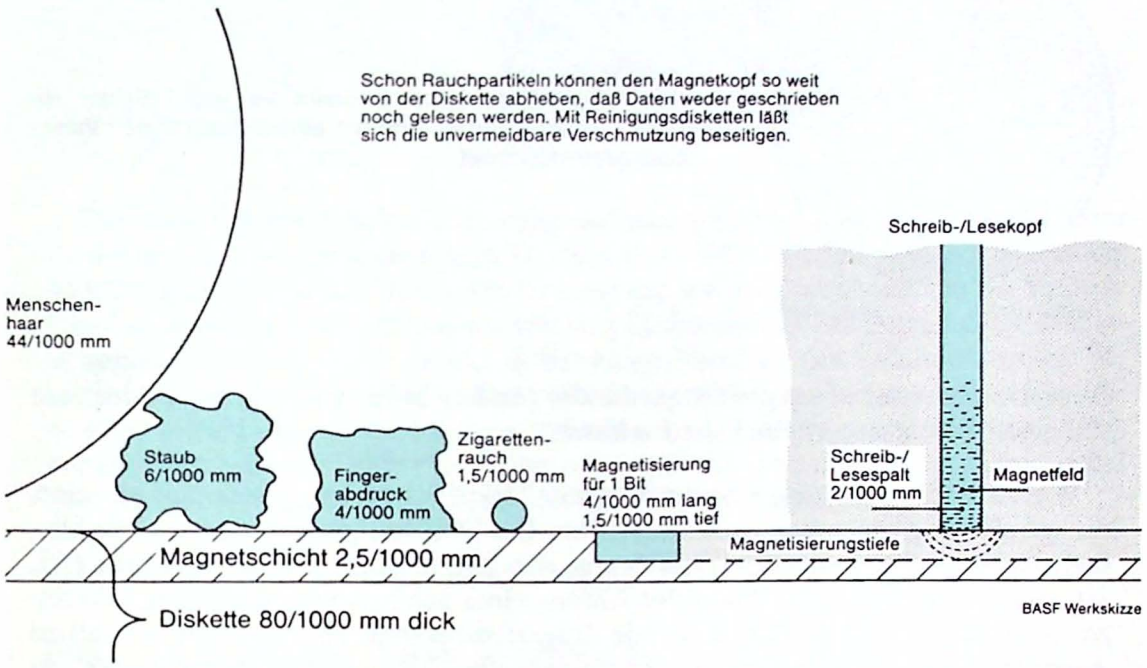


Abb. 1.8.1.2 Größenverhältnisse bei Floppy-Disk-Aufzeichnung

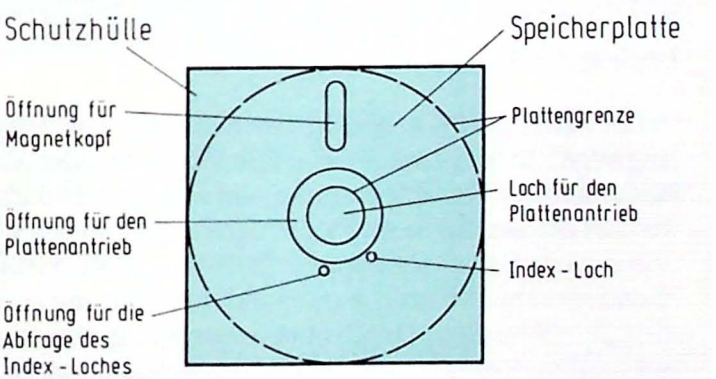


Abb. 1.8.1.3
Aufbau einer „soft-sektorierten" Diskette

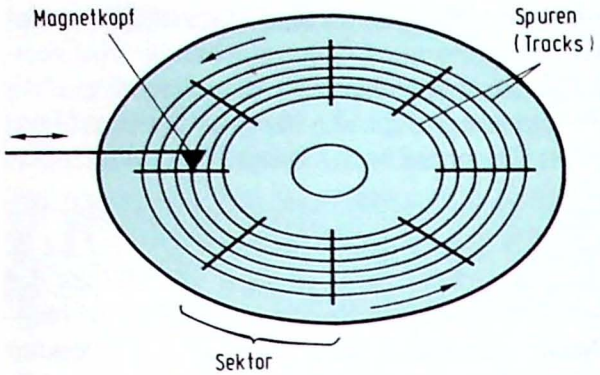


Abb. 1.8.1.4
Die Daten werden in konzentrischen Spuren aufgezeichnet

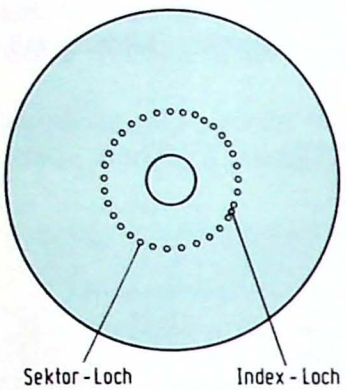


Abb. 1.8.1.5
Hardsektorierte Disketten besitzen für jeden Sektor ein Loch. Der erste Sektor wird mit einem zusätzlichen Index-Loch gekennzeichnet

deren Durchmesser etwas größer als die der runden Trägerfolie ist, so daß letztere von einem Motor angetrieben werden kann.

Wie viele Rillen hat eine Schallplattenseite? Eine Scherzfrage — sie hat nur eine, die spiralförmig von außen nach innen führt. Die Datenaufzeichnung auf einer Diskette geschieht dagegen nicht spiralförmig, sondern in konzentrischen Spuren (35, 40 oder 80, *Abb. 1.8.1.4*). Damit der Anfang einer solchen Spur eindeutig erkennbar ist, weisen die Hülle und die runde Trägerfolie jeweils ein Loch auf, so daß an dieser Stelle Licht durchgelassen wird, wenn beide Löcher genau übereinander stehen. Dieses Indexloch wird im Laufwerk optisch abgetastet. Sogenannte hardsektorierte Disketten (heute kaum noch üblich) besitzen für jeden Sektor ein solches Loch in der Trägerfolie *(Abb. 1.8.1.5)*.

Bei manchen Disketten ist der Mittelteil der Trägerfolie mit einem Kunststoffring verstärkt, um eine präzise Führung durch die Motorachse sicherzustellen, insbesondere bei Diskettengrößen von weniger als 5 Zoll Durchmesser. Übliche Diskettendurchmesser sind 8 Zoll, 5,25 Zoll und 3,5 Zoll. Eine 3,5-Zoll-Disk nimmt ebensoviele Daten auf wie eine 5,25-Zoll-Disk, weil bei ihr die Spurbreite entsprechend verringert wurde. Eine Sonderausführung ist die „Quickdisk" *(Abb. 1.8.1.6)*, die „nur" 64 KByte faßt, wobei wegen der geringen nötigen Präzision der Preis eines Laufwerks recht niedrig gehalten werden konnte.

Abb. 1.8.1.6
Für geringere Kapazitäts-Anforderungen wurde die „Quick Disk" entwickelt

Die Daten werden nicht blockweise auf eine ganze Spur geschrieben. Vielmehr wird jede Spur in Sektoren unterteilt, die z. B. je 256 Datenbytes fassen. Nun ist ein Mikroprozessor unter Umständen zu langsam, um die Datenblöcke in der Reihenfolge auszugeben oder einzulesen, wie sie physikalisch als Sektoren auf der Diskette angeordnet sind. Man wendet daher einen Trick an, der als „Interleaving" bezeichnet wird: Die z. B. 15 Sektoren einer Spur werden in einer anderen Reihenfolge geschrieben, die dem Prozessor zwischen zwei Blöcken genug Zeit zur Verarbeitung läßt, während sich die Diskette weiterdreht. Beim Computer Apple-II mit dem Betriebssystem DOS 3.3 lautet die Sektorfolge beispielsweise:

0, 13, 11, 9, 7, 5, 3, 1, 14, 12, 10, 8, 6, 4, 2, 15

Diese „physikalischen" Sektoren entsprechen den „logischen" Sektoren 0...15. Der Prozessor hat also zur Verarbeitung eines Sektor-Datenblocks jedesmal fast eine ganze Diskettenumdrehung Zeit. Bei anderen Computern und Betriebssystemen ist die Interleaving-Reihenfolge allerdings anders.

Abb. 1.8.1.7 zeigt den mechanischen Aufbau eines 8-Zoll-Floppy-Disk-Laufwerks. Ein Schrittmotor bewegt den Schreib-/Lese-Kopf hin und her, ein weiterer Motor läßt die Diskette rotieren. Für das Indexloch ist eine optische Abtastung eingebaut. Eine zweite Lichtschranke erkennt, wann der Kopf die Spur mit der Nummer 0 (sie ist die äußerste auf der Diskette) erreicht hat.

Eine weitere Lichtschranke ist für die Schreibschutz-Kerbe am Rand der Diskette vorhanden. Diese kann mit einem undurchsichtigen Klebeband überklebt werden,

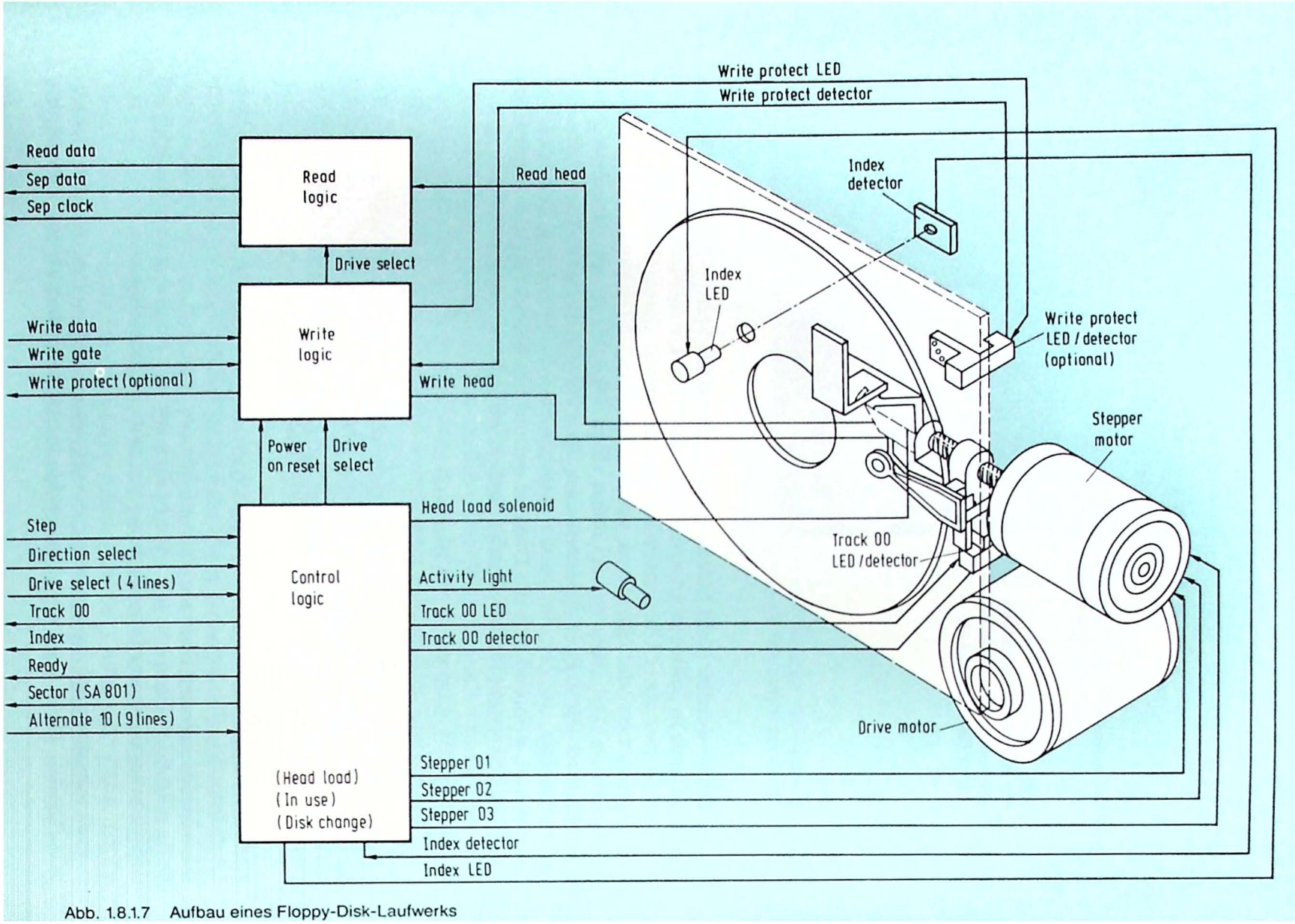


Abb. 1.8.1.7 Aufbau eines Floppy-Disk-Laufwerks

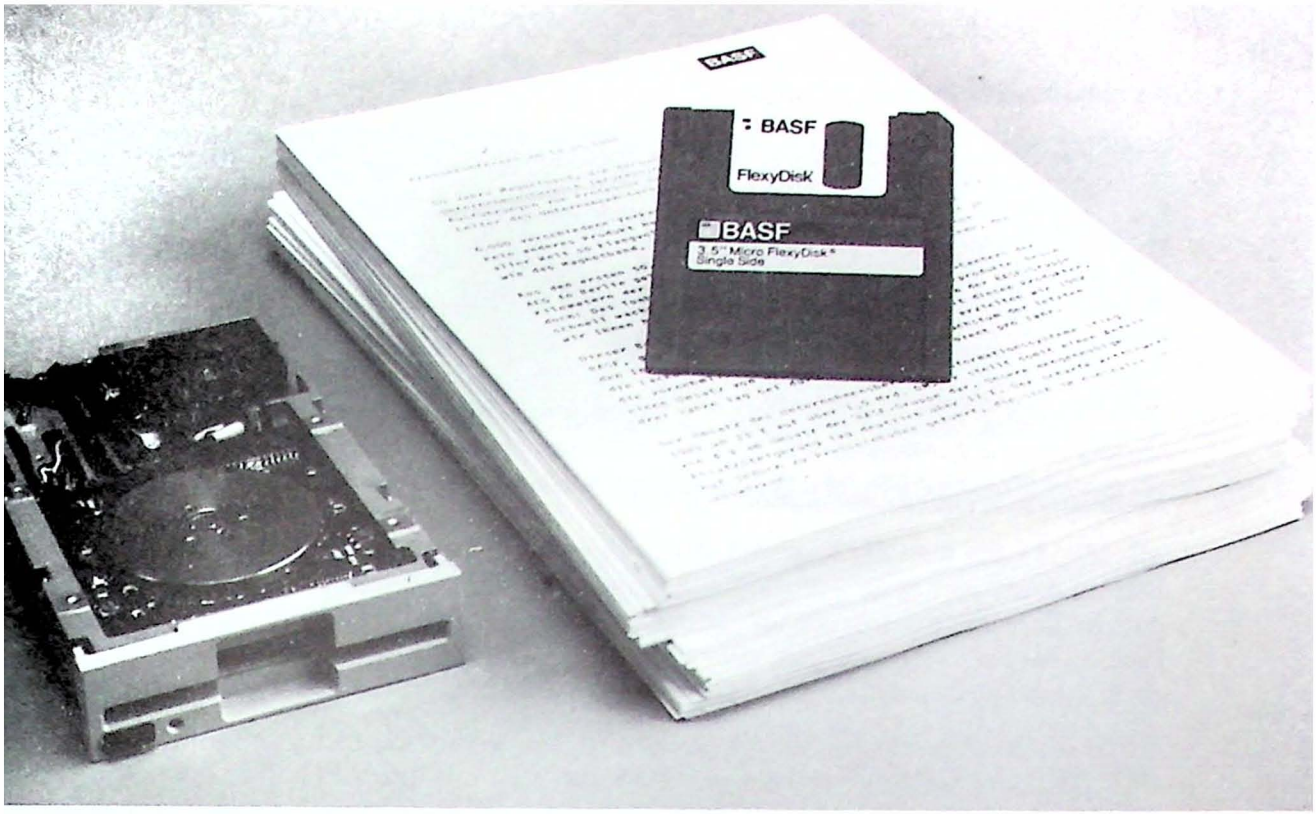


Abb. 1.8.1.8 Vergleich der Größe eines 3,5-Zoll-Laufwerks mit der Menge der damit auf einer Diskette speicherbaren Daten

wenn man verhindern möchte, daß auf diese Diskette geschrieben werden kann. Damit wird auch das versehentliche Löschen oder Verändern von Daten verhindert.

Wie ein 3,5-Zoll-Laufwerk mit immerhin 1 MByte Kapazität aussieht, zeigt *Abb. 1.8.1.8.* Diese Datenmenge entspricht rund 400 DIN-A4-Seiten, die ebenfalls abgebildet sind.

Für die Codierung von 0 und 1 zur Aufzeichnung der einzelnen Bits gibt es sehr unterschiedliche Verfahren; sie werden im folgenden dargestellt.

Frequenzmodulation (FM)

Das FM-Verfahren zeichnet pro Zeiteinheit für jedes 0-Bit nur einen, für jedes 1-Bit hingegen zwei Taktimpulse auf *(Abb. 1.8.1.9).* Dieses Verfahren hat auch den Namen „Single Density" und ist die Standard-Methode für 8-Zoll-Disketten. Beim Lesen kann man z. B. mit einem 3-µs-Monoflop oder einer Phase-Locked-Loop-Schaltung (PLL) aus dem Abstand der Taktimpulse (bzw. umgekehrt proportional, aus der Frequenz) die einzelnen Datenbits rückgewinnen; die nötige Schaltung heißt „Datenseparator". Natürlich muß man am Anfang eines Sektors spezielle, eindeutig erkennbare Synchronisations-Bytes aufzeichnen, damit man später erkennen kann, wo das erste Bit des ersten Datenbytes beginnt. Doch ist das natürlich auch bei allen anderen Aufzeichnungsverfahren nötig.

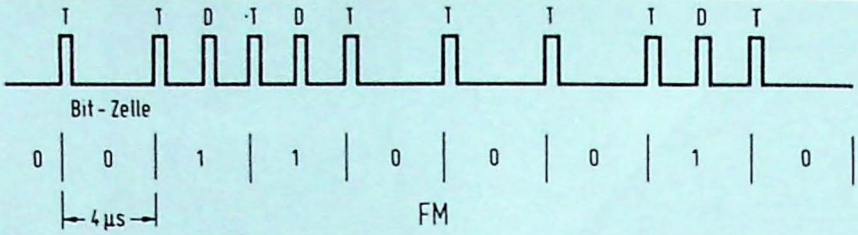


Abb. 1.8.1.9 Aufzeichnung in Frequenzmodulation

Modifizierte Frequenzmodulation (MFM)

Mit MFM kann man die Kapazität einer Floppy verdoppeln *(Abb. 1.8.1.10)*. Jede Bitzelle dauert nur noch 2 µs. Es gilt folgende Regel: Bei einem 1-Bit wird ein Impuls in der Mitte der Bitzelle aufgezeichnet. Bei einem 0-Bit wird am Anfang der Bitzelle ein Puls aufgezeichnet, aber nur dann, wenn das vorhergehende Bit nicht auch 1 war. Die Impulsfolge, die bei lauter Null-Bits übrig bleibt, ist der Takt.

Für die Rückgewinnung der Daten genügt hier natürlich nicht ein Monoflop. *Abb. 1.8.1.11* zeigt das Prinzip: Der Takt kann z.B. mit einer PLL-Schaltung zurückgewonnen werden. Das Verfahren ist u.a. gegen Schwankungen der Rotationsgeschwindigkeit etwas empfindlicher als FM, bei der heute erreichbaren Präzision stellt das aber kein Problem dar. Bei 5,25-Zoll-Laufwerken wird deshalb heute überwiegend MFM eingesetzt.

Wegen der endlichen Größe der Magnetschicht-Partikel und der Spaltbreite des Magnetkopfes entfernen sich zwei Impulse beim Lesen scheinbar voneinander.

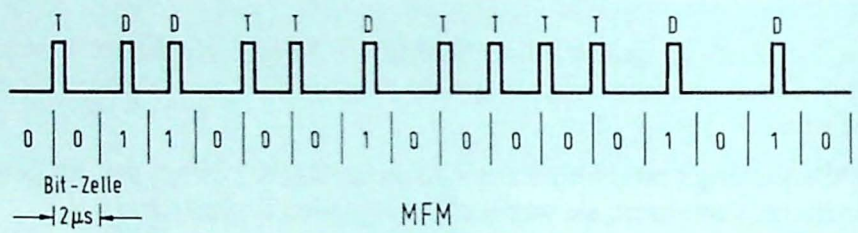


Abb. 1.8.1.10 Aufzeichnung mit doppelter Dichte (MFM)

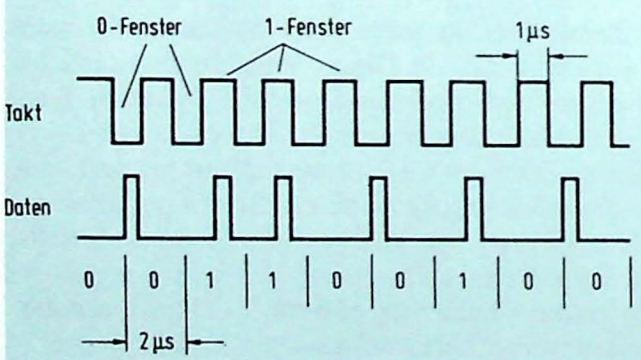


Abb. 1.8.1.11
Decodieren von MFM-Daten

Um diesen Effekt zu kompensieren, rückt man sie schon bei der Aufzeichnung um etwa 150 ns näher zusammen, was man Precompensation nennt.

M²FM-Aufzeichnung

Bei M²FM *(Abb. 1.8.1.12)* wird wie bei MFM für 1-Bits in der Mitte einer Bitzelle ein Impuls aufgezeichnet. Bei 0-Bits wird nur dann am Anfang einer Bitzelle ein Impuls aufgezeichnet, wenn die vorhergehende Zelle (ob sie nun 1 oder 0 war) keinen Puls enthielt. Eine Precompensation ist hier meist nicht nötig, da der Abstand der Impulse prinzipiell größer als bei MFM ist. Das Wiedereinlesen solcher Daten ist aber erheblich aufwendiger. Deshalb konnte sich dieses trickreiche Verfahren in der Praxis nicht durchsetzen.

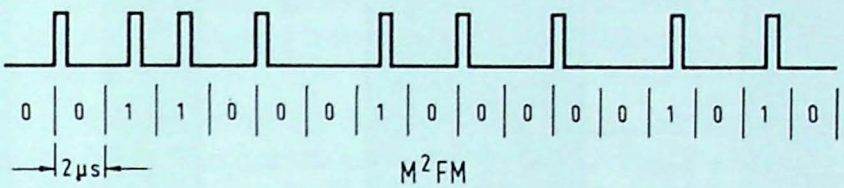


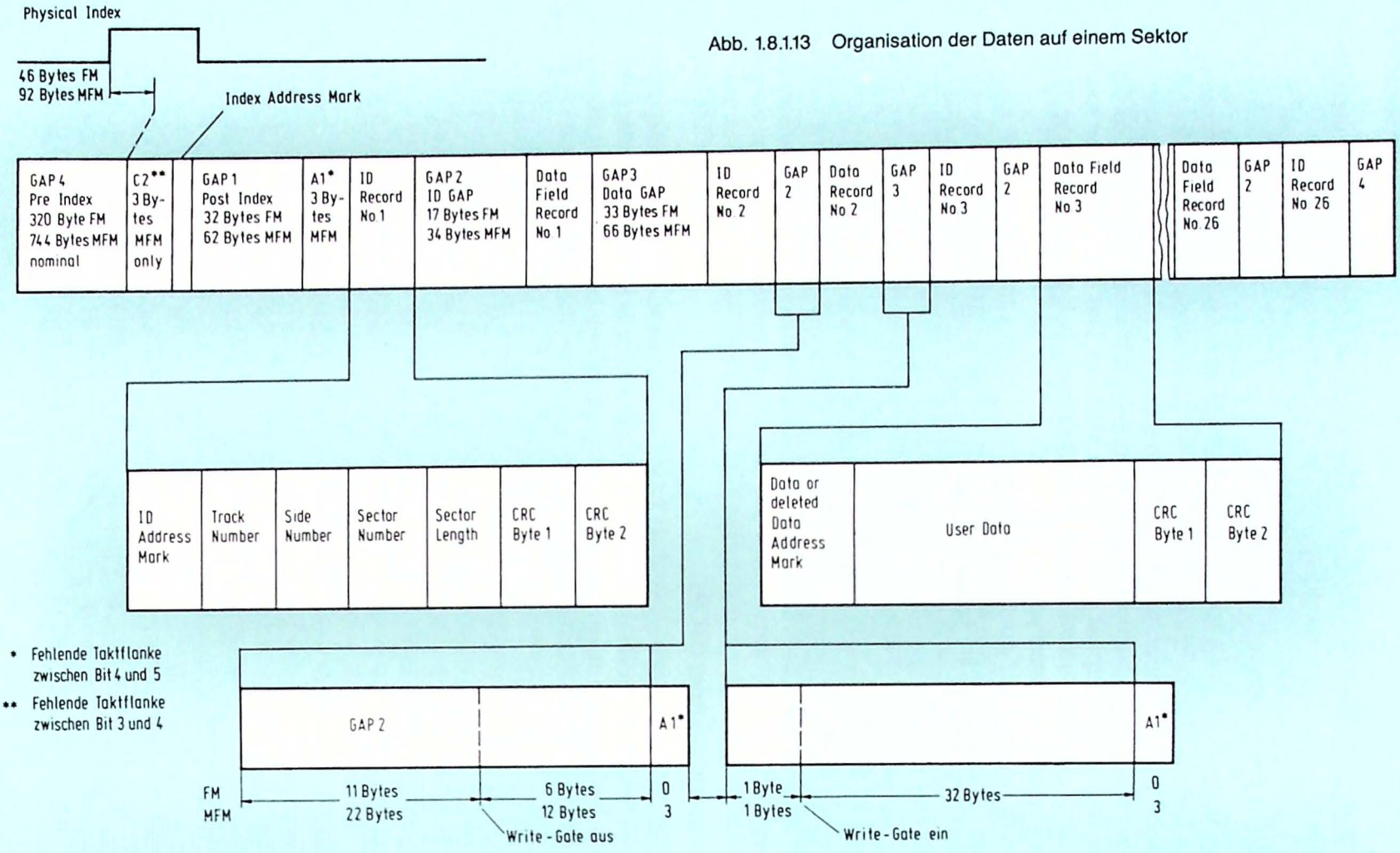
Abb. 1.8.1.12 Prinzip der M²FM-Aufzeichnung

Gruppencodierung (GCR)

Group Code Recording (GCR) wird beispielsweise bei Apple-Computern eingesetzt. Je sechs Datenbits werden über eine Tabelle in acht Aufzeichnungsbits umgesetzt, so daß nie mehr als zwei aufeinanderfolgende Null-Bits vorkommen, da sonst die Taktinformation verloren ginge. Diese Tabelle lautet (Datenbyte, aufzuzeichnendes Byte, jeweils hexadezimal):

00 = 96	10 = B4	20 = D6	30 = ED
01 = 97	11 = B5	21 = D7	31 = EE
02 = 9A	12 = B6	22 = D9	32 = EF
03 = 9B	13 = B7	23 = DA	33 = F2
04 = 9D	14 = B9	24 = DB	34 = F3
05 = 9E	15 = BA	25 = DC	35 = F4
06 = 9F	16 = BB	26 = DD	36 = F5
07 = A6	17 = BC	27 = DE	37 = F6
08 = A7	18 = BD	28 = DF	38 = F7
09 = AB	19 = BE	29 = E5	39 = F9
0A = AC	1A = BF	2A = E6	3A = FA
0B = AD	1B = CB	2B = E7	3B = FB
0C = AE	1C = CD	2C = E9	3C = FC
0D = AF	1D = CE	2D = EA	3D = FD
0E = B2	1E = CF	2E = EB	3E = FE
0F = B3	1F = D3	2F = EC	3F = FF

Abb. 1.8.1.13 Organisation der Daten auf einem Sektor



Die 256 Bytes eines Sektors werden also zuerst in 342 Bytes mit je zwei führenden Nullen aufgespalten und anschließend nach obenstehender Tabelle umgerechnet und abgespeichert.

Das Schreiben der Bits kann bei GCR sehr einfach erfolgen: Für ein 1-Bit wird ein Taktimpuls in eine Bitzelle (als Zeiteinheit) geschrieben, für ein 0-Bit dagegen keiner. Die Art der Codierung stellt sicher, daß keine größeren Taktimpuls-Lücken auftreten, so daß eine Taktrückgewinnung keine Probleme verursacht.

Markierung der Datenfelder

Leider kann man die aufzuzeichnenden Daten nicht einfach ohne zusätzliche Informationen auf die Diskette schreiben. Zum Beispiel sind bei sog. softsektorierten Disketten, die nur ein einziges Indexloch besitzen, Markierungen für den Anfang eines jeden Sektors nötig.

Abb. 1.8.1.13 zeigt die bei FM-Aufzeichnung (Single Density) gebräuchlichen Markierungen. Nach dem optisch erkannten Indexloch folgt im Abstand von etwa 46 Byte die Index-Adreßmarke. Prinzipiell wäre es damit sogar möglich, auf die optische Erkennung des Indexlochs ganz zu verzichten; beim Apple-II wird der Spuranfang tatsächlich nur mit der Index-Adreßmarke festgestellt. Nach ihr folgen 32 Füllbytes und dann die sogenannte ID-Adreßmarke. Sie kündigt den Beginn der Spur- und Sektor-Information an.

Im folgenden Feld stehen die Spur- und Sektornummer, die Seite der Diskette und die Zahl der Bytes im nachfolgenden Datenfeld. Der Controller kann also feststellen, ob der Laufwerkskopf wirklich auf der richtigen Spur positioniert wurde.

Die CRC-Bytes dienen der Fehlererkennung (Cyclic Redundancy Check = zyklische Redundanz-Prüfung). Im Gegensatz zur Prüfsummenbildung werden nicht einfach alle Datenbytes addiert. Vielmehr wird ein Prüfwort gebildet, das so errechnet ist, daß sich die Gesamt-Bitfolge nun vollständig (d.h. ohne Rest) durch ein vorher definiertes „Generator-Polynom" teilen läßt. Das meistverwendete Polynom hierfür lautet:

$$G(x) = 1 + x^5 + x^{12} + x^{16}$$

Es wurde vom CCITT genormt und ermöglicht eine sehr zuverlässige Fehlererkennung. Im Gegensatz zur Prüfsummen-Methode werden damit z.B. auch vertauschte Bytes erkannt. Die CRC-Bildung erfolgt normalerweise automatisch im Floppy-Disk-Controller (FDC), so daß darauf nicht näher eingegangen werden muß.

Auch das Datenfeld, das auf die Daten-Adreßmarke folgt, wird mit CRC-Bytes abgeschlossen. Die einzelnen Sektoren sind voneinander durch eine Reihe von Füllbytes getrennt. Eine weitere Sorte von Adreßmarke dient der Kennzeichnung gelöschter Datenblöcke (Deleted Data Address Mark).

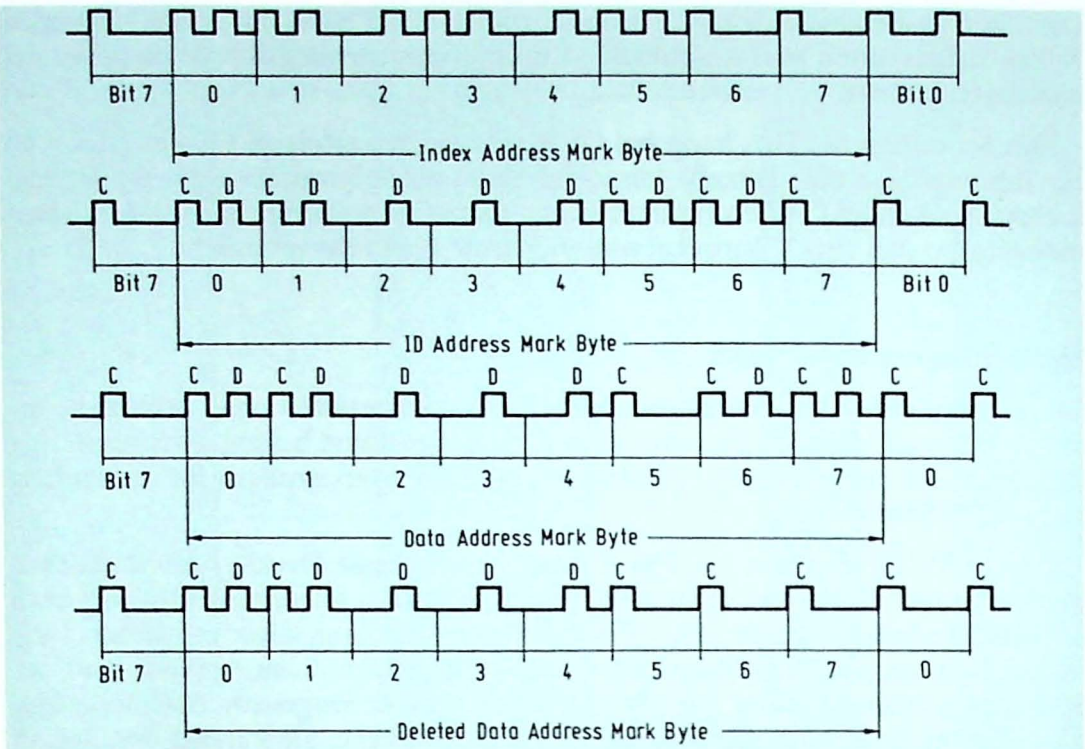


Abb. 1.8.1.14 Die Codierungen der unterschiedlichen Adreßmarken

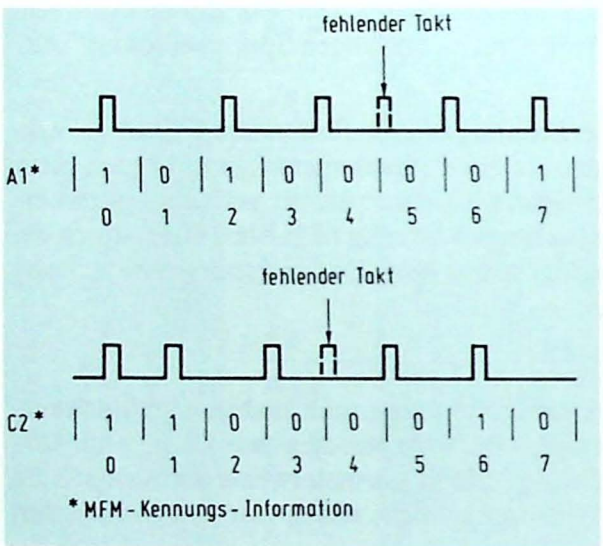


Abb. 1.8.1.15
Adreßmarken-Codierung bei MFM-Aufzeichnung

Abb. 1.8.1.14 macht den Aufbau der Adreßmarken bei FM-Aufzeichnung deutlich. Der Trick dabei ist, daß diese Bitmuster bei Datenbytes so nie vorkommen können, so daß der Controller stets zuverlässig den Sektorbeginn finden kann. Dies wird durch gezieltes Weglassen von Taktimpulsen erreicht. Bei MFM (Double

Density), wo ja ohnehin wegen der dichteren Aufzeichnung schon Taktimpulse fehlen, werden die beiden Bitmuster in *Abb. 1.8.1.15* verwendet, bei denen nochmals Impulse weggelassen werden. Der Aufbau der Adreßmarken bei MFM lautet dann:

C2	C2	C2	FC	Index-Adreßmarke
A1	A1	A1	FE	ID-Adreßmarke
A1	A1	A1	FB	Daten-Adreßmarke

FB, FC und FE werden dabei jeweils mit Taktimpulsen aufgezeichnet.

Die ID-Adreßmarke gibt auch an, wie viele Datenbytes in einem Sektor stehen. Hierfür gibt es vier Möglichkeiten:

ID-Feld	Sektorlänge/Byte
00	128
01	256
02	512
03	1024

Das Formatieren

Bevor erstmals Daten auf eine frische Diskette geschrieben werden können, muß sie erst formatiert werden. Dabei werden alle Marken und „Lückenfüller" auf die Diskette geschrieben. Hierzu gibt es beim Floppy-Disk-Controller spezielle Steuerbefehle.

Leider gibt es beim Formatieren ganz erhebliche Unterschiede zwischen den einzelnen Computer-Herstellern. Selbst die Tatsache, daß zwei Computer dasselbe Betriebssystem (z. B. CP/M) verwenden, ist noch keine Gewähr dafür, daß die Disketten auch tatsächlich austauschbar sind. Lediglich bei 8-Zoll-Disketten hielt man sich meist an das Format IBM-3740 (FM) oder IBM-34 (MFM). Für 5,25-Zoll-Floppies gibt es zwar einen Standard namens ECMA-70, er wird aber wegen seiner geringen Aufzeichnungsdichte kaum eingesetzt, und die Computerhersteller haben meist ihr Privat-Format erfunden. Die Anzahl der Sektoren pro Spur variiert dabei von 5 bis 18, pro Diskettenseite werden 35, 40, 77 oder 80 Spuren aufgezeichnet. Bei MFM-Systemen wird manchmal die Spur 0 in FM geschrieben (z. B. bei ECMA-70 und IBM-34).

1.8.2 Festplatten (Hard Disks)

Bei Floppy-Disk-Laufwerken liegt der Schreib-/Lese-Kopf unmittelbar auf der magnetischen Beschichtung der Floppy auf. Die vertretbare Rotationsgeschwindigkeit und damit auch die Geschwindigkeit des Datenzugriffs ist hierdurch begrenzt. Ebenso kann der Kopf nicht beliebig genau positioniert werden, weil das Träger-

material nicht starr, sondern biegsam ist. Das begrenzt wiederum die Zahl der Spuren auf der Scheibe und somit die Speicherkapazität.

Die sogenannten Festplatten heißen so, weil sie eine starre Form aufweisen. Sie schleifen nicht innerhalb einer Schutzhülle, sondern sind in einem entsprechend konstruierten Laufwerk staubdicht untergebracht. Der Austausch der runden, magnetisch beschichteten Platte ist konstruktiv entweder gar nicht oder nur in Ausnahmefällen vorgesehen. Eine Festplatte dient also normalerweise als Massenspeicher, aber nicht zum Datenträger-Austausch z. B. zum Laden gekaufter Software.

Drei Unterschiede kennzeichnen die Festplatte gegenüber einer Floppy Disk:

1. Der Schreib-/Lese-Kopf berührt im Betrieb die Platte nicht, weil er durch das wegen der Rotation automatisch entstehende Luftpolster in geringem Abstand über ihr gehalten wird (Flughöhe < 1 µm).
2. Die Umdrehungsgeschwindigkeit kann bei Festplatten recht hoch sein, da eine Reibung weder mit einer Schutzhülle noch mit dem Kopf auftritt. Dadurch ergibt sich eine hohe Daten-Transferrate.
3. Da die Festplatte kein Austausch-Datenträger ist, läßt sich eine hohe Zentriergenauigkeit und damit eine hohe Anzahl von Spuren bzw. eine hohe Speicherkapazität erzielen (10...150 MByte bei 5,25 Zoll Durchmesser).

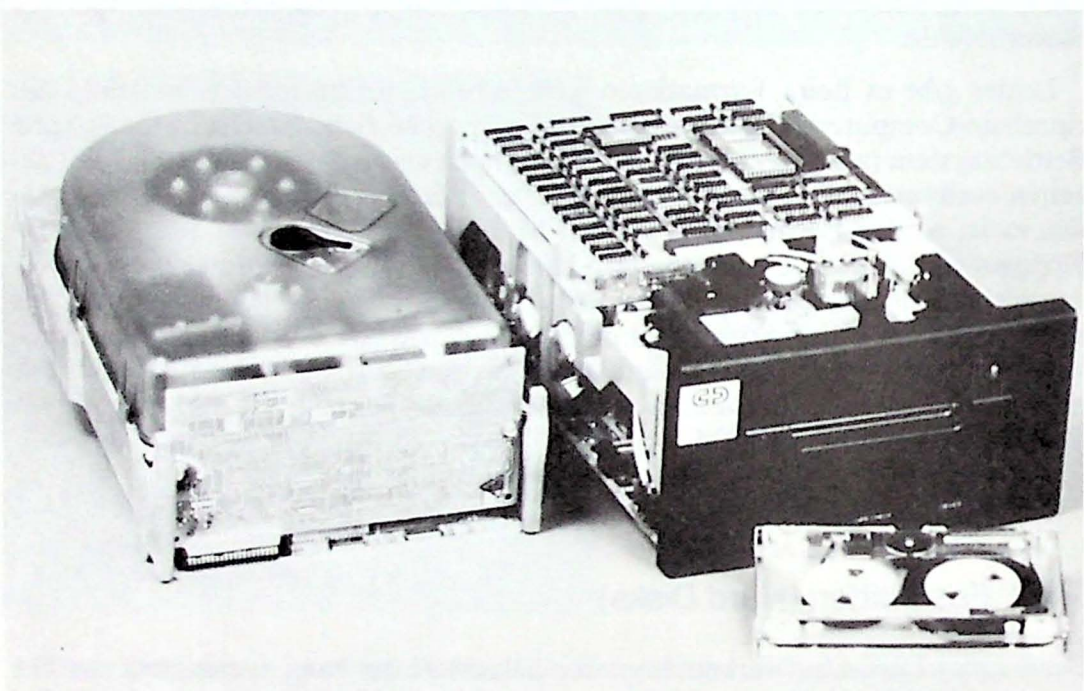


Abb. 1.8.2.1 Festplatten-Laufwerk (Hard Disk). Die Innereien sind unter einer Staubschutz-Haube untergebracht. Rechts ein passender Magnetband-Streamer

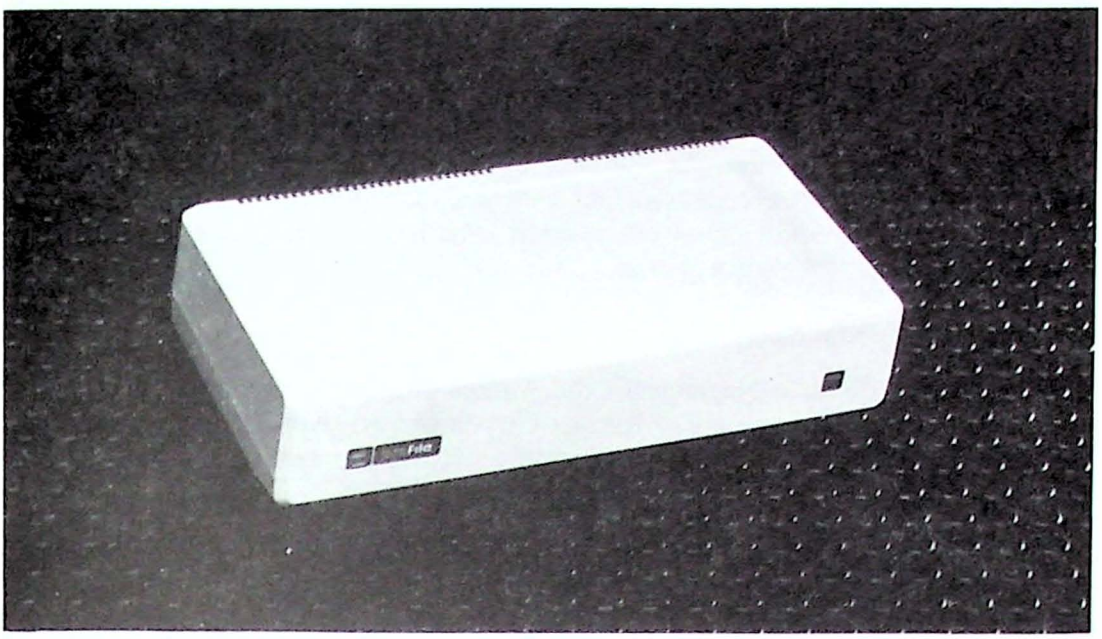


Abb. 1.8.2.2 Profile-Festplatte von Apple mit 5 MByte Kapazität

Die Aufzeichnung selbst erfolgt prinzipiell wie bei einer herkömmlichen Floppy, allerdings mit einer höheren Datentransferrate. Weil sich der Datenträger bei einer Festplatte nicht so leicht austauschen läßt, stellt das Anfertigen einer Sicherheits-Kopie der gespeicherten Daten ein gewisses Problem dar. Hier gibt es zwei Lösungen. Die eine ist, man fügt dem Festplatten-Laufwerk ein Floppy-Laufwerk bei. Wenn man allerdings den gesamten Inhalt einer 10-MByte-Festplatte auf 500-KByte-Floppies sichern möchte, so bräuchte man hierzu immerhin 20 Floppies!

Die andere Möglichkeit ist ein sogenannter Streamer. Das ist praktisch ein Kassettenlaufwerk hoher Aufzeichnungsdichte. Hier genügt eine Datenbandkassette, um den gesamten Inhalt der Festplatte in einigen Minuten auf die Kassette zu überspielen. *Abb. 1.8.2.1* zeigt links ein Festplatten-Laufwerk und rechts einen passenden Streamer von Control Data. Er arbeitet hier mit 55 KBit/s und zeichnet auf dem Band parallel 11 Spuren auf. Das Band läuft mit 55 Zoll/s. 40 MByte werden in 19 Minuten überspielt.

Abb. 1.8.2.2 zeigt ein Festplatten-Laufwerk im Gehäuse (Apple-Profile). Es ist leicht zu erkennen, daß tatsächlich keine Öffnung zum Austausch der Platte da ist. Auf Floppy gelieferte Programme müssen daher entweder direkt vom Floppy-Laufwerk geladen oder zunächst auf die Festplatte kopiert werden. Es versteht sich von selbst, daß für den Betrieb einer solchen Platte ein geeigneter Controller sowie die nötige Steuersoftware vorhanden sein müssen, um die großen Datenmengen effektiv verwalten zu können.

1.8.3 Kassettenaufzeichnung

Floppy-Disk-Laufwerke kosten einige hundert DM. Auch wenn sie zahlreiche Vorteile bieten, kommen sie als preiswertes Speichermedium z.B. für Heimcomputer nur in zweiter Linie in Betracht. Zu Beginn der Mikrocomputer-Ära spielte die Daten- bzw. Programmaufzeichnung auf Tonbandkassetten eine noch größere Rolle als heute, weil Floppies zum einen noch sehr teuer waren und zum anderen kaum geeignete Treiber-Software für sie zur Verfügung stand.

Der Kansas-City-Standard

Eine der ersten de-facto-Normen für die Aufzeichnung auf handelsübliche Compactcassetten war der sogenannte Kansas-City-Standard. Auf ihn einigten sich am 7. und 8. November 1975 in Kansas City die wichtigsten amerikanischen Mikrocomputer-Hersteller auf Einladung der Zeitschrift „Byte".

Die Daten werden in Frequenzmodulation (FM) codiert. Ein 1-Bit entspricht acht Schwingungen eines 2400-Hz-Tons, ein 0-Bit dagegen vier 1200-Hz-Schwingungen. Daraus ergibt sich eine Geschwindigkeit von 300 Baud *(Abb. 1.8.3.1)*. Den acht Datenbits eines Bytes wird (wie beim seriellen Format an einer V.24-Schnittstelle) ein Startbit voraus- und ein Stopbit nachgestellt. Pro Sekunde können 30 Bytes aufgezeichnet werden. Da es für einen Mikroprozessor kein Problem darstellt, die beiden nötigen Tonfrequenzen zu erzeugen und auch zu decodieren, beschränkt sich die Hardware auf einen Eingangs-Verstärker, der das Niederfrequenz-Signal des Kassettenrecorders auf TTL-Pegel bringt.

Der Kansas-City-Standard wurde allerdings von vielen Computer-Herstellern schnell wieder über Bord geworfen, als sich herausstellte, daß er wegen der geringen Geschwindigkeit für längere Programme und Datensätze kaum zumutbar wäre. Zudem normte er ohnehin nur den physikalischen Teil der Aufzeichnung, nicht aber die Reihenfolge der Bits und Bytes auf der Kassette.

Die Folge war ein sich nun entwickelnder Wildwuchs von unterschiedlichsten Aufzeichnungsformaten, meist mit deutlich höherer Geschwindigkeit als die 300 Baud des Kansas-City-Standards. So wurde versucht, den Standard unter Beibehaltung der beiden Frequenzen auf 1200 Bd „aufzubohren": Ein 0-Bit entspricht dann einer einzigen 1200-Hz-Schwingung und ein 1-Bit zwei 2400-Hz-Schwingungen. Leider stellte sich heraus, daß dieses Verfahren Probleme bei nicht geradlinigem Frequenzgang des Recorders machte.

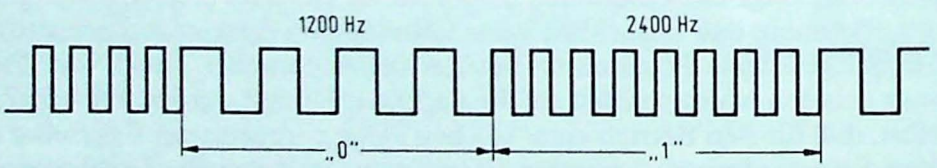
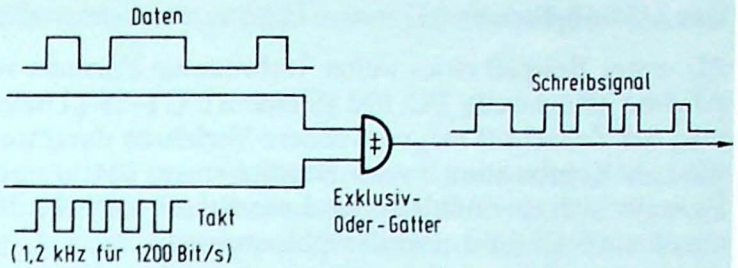


Abb. 1.8.3.1 Kassettenaufzeichnung nach dem Kansas-City-Standard

Abb. 1.8.3.2
Gewinnung eines
phasencodierten Signals



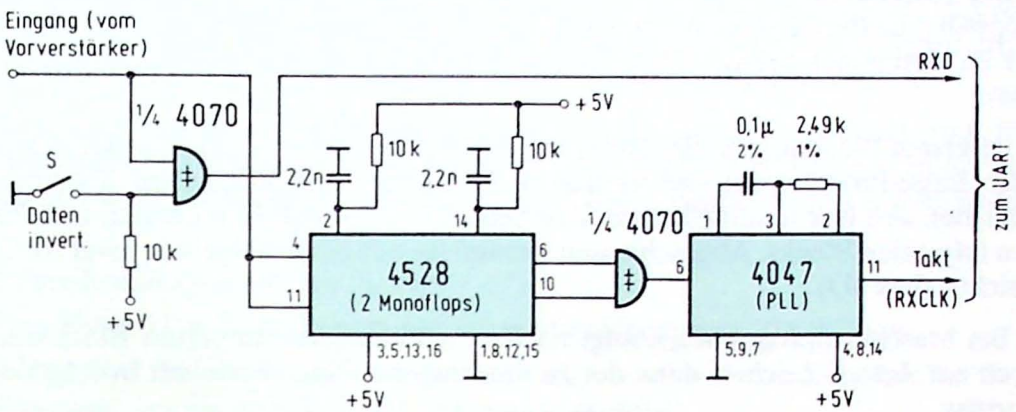


Abb. 1.8.3.3 Schaltung zur Rückgewinnung der Daten aus einer phasencodierten Aufzeichnung

Die Phasen-Codierung

Ein recht beliebtes Verfahren ist das sogenannte „Phase Encoding" (Phasencodierung, *Abb. 1.8.3.2*). Hierbei wird das Daten- mit einem der Baudrate entsprechenden Taktsignal an einem Exklusiv-Oder-Gatter verknüpft. Somit ist sichergestellt, daß auch dann ein Polaritätswechsel auftritt, wenn eine Folge von Bits nur Null oder nur Eins ist. (Bekanntlich können auf einem Tonband keine Gleich-, sondern nur Wechselspannungen aufgezeichnet werden.) Bei einem Wechsel des Bitwerts entsteht so bei der aufgezeichneten Taktfrequenz ein Phasensprung von 180 Grad (vgl. MFM bei Floppy-Disk-Aufzeichnung!). Ein besonderer Vorteil dieses PE-Verfahrens ist, daß die Bandbreite des Kassettenrecorders, in der er Signale ohne größere Phasenfehler übertragen kann, nur wenig größer als die Taktfrequenz sein muß. So ist es möglich, auf einem handelsüblichen Billig-Recorder Daten mit 2400 Baud aufzuzeichnen, d.h. 240 Zeichen bzw. Bytes pro Sekunde. Ein gewisser Aufwand ist beim PE-Verfahren allerdings für die wiedergabeseitige Rückgewinnung des Takts erforderlich. Die Schaltung in *Abb. 1.8.3.3* verwendet hierzu eine Phase-Locked-Loop-Schaltung, deren Oszillator auf der Taktfrequenz rastet.

Ein ähnliches Verfahren nennt man NRZ (No Return to Zero), weil auch bei aufeinanderfolgenden Null-Bits trotzdem Flußwechsel der Magnetisierung erzwungen werden.

Das AIM-65-Format

Als erstes Beispiel eines weiter verbreiteten Formats sei das bei den Computern AIM-65 (Rockwell), PC-100 (Siemens), CT-65 (Thaler) und MC-65 (Bauanleitung der Zeitschrift mc) verwendete Verfahren dargestellt *(Abb. 1.8.3.4)*. Jedes Bit wird als Kombination zweier Schwingungen (2400 und 1200 Hz) aufgezeichnet. Es ergibt sich eine mittlere Geschwindigkeit von etwa 1000 Baud. Jedes Byte wird mit einem Start- und einem Stopbit ergänzt.

Die Aufzeichnung beginnt mit 32 SYN-Zeichen (hex 16), gefolgt von einem Doppelkreuz (hex 23) zur Markierung des Datenbeginns. Dann folgt ein Byte mit der Blocknummer, beginnend bei 00 (die Aufzeichnung erfolgt in einzelnen Blökken).

Im ersten Block folgen fünf Bytes mit einem File-Namen (ASCII). Bei Textfiles oder Basic-Programmen (sie werden in Form von ASCII-Zeichen übertragen) schließen sich hier unmittelbar die Zeichen an — maximal 74 im ersten und 79 in den folgenden Blocks. Abgeschlossen werden sie mit einer Folge von zwei Return-Zeichen (hex 0D).

Bei Maschinenprogrammen folgt nach dem Filenamen im ersten Block zuerst noch ein Return-Zeichen, dann der zu übertragende Speicherinhalt in folgendem Format:

;NNAHALD1D2...DNCHCL
;NNAHALD1D2...DNCHCL
usw.
;00MHMLMHML

Hierbei bedeuten: NN = Zahl der Datenbytes in der Zeile, AH = höherwertiges Adressenbyte, AL = niederwertiges Adressenbyte, D1...DN = Datenbytes, CH = höherwertiger Prüfsummenteil, CL = niederwertiger Prüfsummenteil, MH/ML = Zahl der insgesamt übertragenen Zeilen (inklusive der letzten). Jede Zeile beginnt mit einem Strichpunkt (ASCII 3B) und endet mit einem Return-Zeichen. Dieses Format ist auch als 6502-Hex-Format bekannt. Es ermöglicht, innerhalb einer einzigen Aufzeichnung mehrere, nicht zusammenhängende Adressenbereiche zu übertragen.

Am Ende jedes Blocks wird eine 2-Byte-Prüfsumme mit aufgezeichnet, die die Erkennung von Fehlern ermöglicht. Ein eventuell verbleibender Freiraum im letzten Block wird mit Null-Bytes aufgefüllt. Das AIM-65-Format hat sich auch mit

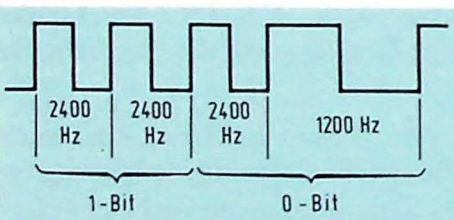


Abb. 1.8.3.4
Aufzeichnung im AIM-65-Format

preiswerten Kassettenrecordern trotz seiner recht hohen Geschwindigkeit als sehr zuverlässig erwiesen.

Das Commodore-Format

Commodore hat bei seinen Computern ein Aufzeichnungsformat vorgesehen, das mit insgesamt drei unterschiedlichen Frequenzen arbeitet *(Abb. 1.8.3.5)*, die mit „Byte", „Long" und „Short" bezeichnet werden. Die Kennzeichnung eines Byteanfangs erfolgt im Gegensatz zu den bisher besprochenen Formaten nicht mit einem Startbit (log. 0), sondern mit einer besonders reservierten Frequenz-Kombination (Byte-Long). Es ist daher auch kein Stopbit nach jedem Byte erforderlich. Allerdings wird jedem Byte ein neuntes Bit zur Paritätsprüfung angehängt.

Abb. 1.8.3.6 macht deutlich, was nun alles auf Kassette aufgezeichnet wird. Ein 10-Sekunden-Vorspann sorgt dafür, daß bei zurückgespulter Kassette trotz Vorspann-Band nichts verloren geht. Der Header-Block enthält folgende Informationen:

1. 9 Synchronisations-Bytes (hex 89...81)
2. Flag-Byte: 01 = Basic-Programm, 02 = Maschinenprogr., 04 = Daten
3. 4 Bytes mit Anfangs- und Endadresse
4. Filename (mit Leerräumen auf 128 Byte aufgefüllt)
5. Prüfsummen-Byte

Nach einer 50-ms-Pause wird der komplette Header wiederholt, für den Fall, daß er beim ersten Mal nicht korrekt gelesen werden konnte.

Der Programmblock (bzw. Datenfile-Block) ist wie folgt aufgebaut:

1. 9 Synchronisations-Bytes (hex 89...81)
2. Datenbytes (codiertes Basic-Programm, nur Textfiles in ASCII)
3. Prüfsumme

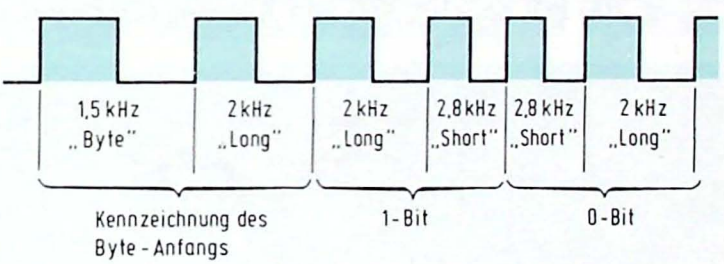


Abb. 1.8.3.5
Das Commodore-Aufzeichnungsformat

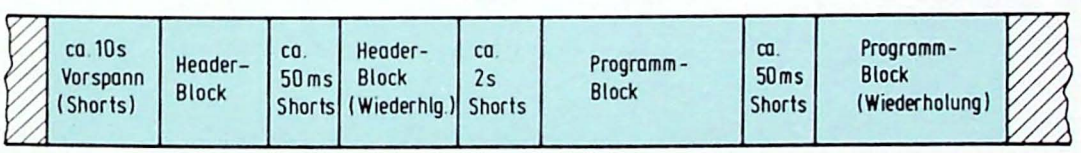


Abb. 1.8.3.6 Commodore-Rechner zeichnen jeden Block sicherheitshalber zweimal hintereinander auf

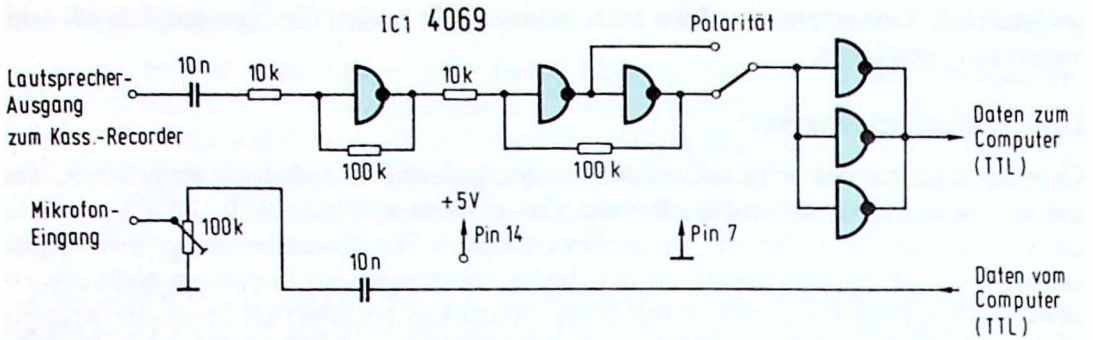


Abb. 1.8.3.7 Mit dieser Schaltung ist es möglich, auch bei solchen Computern herkömmliche Kassettenrecorder zu verwenden, die am Eingang TTL-Pegel erwarten

Auch der Programmblock wird zur Sicherheit zweimal nacheinander aufgezeichnet. Obwohl sich physikalisch eine Geschwindigkeit von knapp 1200 Bd ergibt, muß man wegen der zweimaligen Aufzeichnung daher mit nur 60 Bytes pro Sekunde rechnen (entsprechend 600 Bit/s, weil jedes Byte aus 10 Zellen besteht).

Der Kassettenrecorder-Anschluß der Commodore-Computer (VC-20, C-16, C-64, CBM usw.) ist nicht für herkömmliche Recorder, sondern für eine Sonderversion mit TTL-Ein- und Ausgangspegeln dimensioniert. Will man einen normalen Recorder benutzen, so kann man die Schaltung in *Abb. 1.8.3.7* verwenden. Die richtige Stellung des eingezeichneten Umschalters hängt davon ab, ob der Recorder die Polarität der Daten invertiert oder nicht.

Anschlüsse von Kassettenrecordern

Bei Kassettenrecordern sind ebenso wie bei Spulentonbandgeräten im wesentlichen drei Arten von Anschlüssen verbreitet, nämlich Klinken-, DIN-Mono- und DIN-Stereo-Buchsen *(Abb. 1.8.3.8)*.

Bei Klinkenbuchsen ist für Eingang (Mikrofon, ca. 5 mV an 1 kΩ, oder Auxiliary, ca. 200 mV an >100 kΩ) und Ausgang (Ohrhörer, je nach Lautstärke-Einstel-

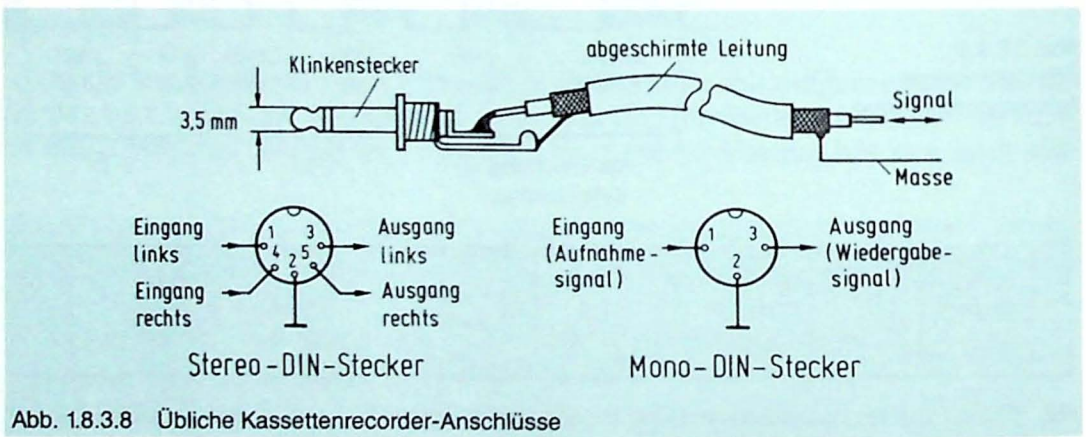


Abb. 1.8.3.8 Übliche Kassettenrecorder-Anschlüsse

lung etwa 1 V an 8 Ω) je ein 3,5-mm-Stecker nötig. Ein 2,5-mm-Stecker, falls vorhanden, dient der Fernsteuerung: Mit einem Relais kann man den Recorder vorübergehend anhalten. Vorsicht: Der Fernsteuer-Anschluß (Remote) ist meist nicht potentialfrei, d.h. der Schaltkontakt liegt nicht an Masse! Eine Potentialtrennung mit einem Relais ist daher auf jeden Fall empfehlenswert.

Bei Verwendung der Fernsteuerung ist auch zu beachten, daß sie meist nicht nur den Motor, sondern auch den Verstärkerteil schaltet. Es ist also nach dem Wiedereinschalten nicht nur die Hochlaufzeit des Motors bis zum Erreichen der normalen Bandgeschwindigkeit (4,75 cm/s), sondern auch eine eventuelle Einschaltverzögerung des Verstärkers zu berücksichtigen, z.B. vor dem Beginn der Aufzeichnung des nächsten Datenblocks.

Bei DIN-Anschlüssen ist oft nicht der Lautsprecher- bzw. Ohrhörer-Ausgang herausgeführt, sondern ein von der jeweiligen Lautstärke-Einstellung unabhängiger Vorverstärker-Ausgang, der bei voller Aussteuerung etwa 700 mV an 600 Ω liefert. Leider ist insbesondere bei preiswerten Recordern auf eine einheitliche Buchsenbelegung kein Verlaß.

1.8.4 Der Strichcode

Über Strichcode-Leser war schon im Kapitel über Sensoren kurz die Rede; *Abb. 1.8.4.1* zeigt den prinzipiellen Aufbau eines solchen optischen Lesers. Es gibt auch Strichcode-Leser, die mit einem Laserstrahl arbeiten, um auch aus Entfernungen bis zu einem Meter z.B. an einer Kaufhauskasse den Code auf Verpackungen noch lesen zu können.

Erzeugt wird der Strichcode mit grafikfähigen Matrix- oder Tintenstrahl-Druckern, aber auch mit Typenrad- bzw. Typenhebel-Druckern, bei denen die Zeichen „dünner Strich" und „dicker Strich" zusätzlich vorhanden sind. Die Vervielfälti-

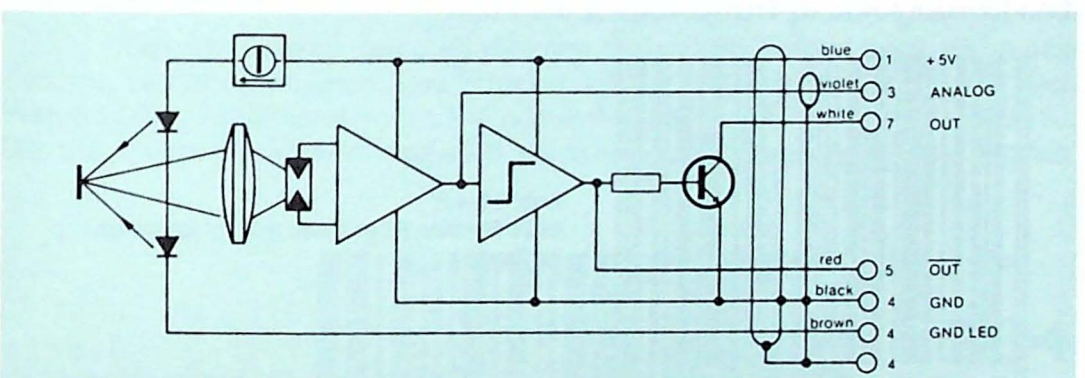


Abb. 1.8.4.1 Innenaufbau eines Strichcode-Lesers mit Reflex-Lichtschranke

gung kann im Offset-, Buch- oder Tiefdruckverfahren erfolgen; allerdings sind dabei die Toleranzen der Strichbreiten einzuhalten.

Im folgenden werden wir uns einige Strichcode-Arten näher ansehen; es sind beileibe nicht alle, da es weltweit viele Standards dafür gibt. In Europa hat sich der sogenannte EAN-Code weit verbreitet.

Der EAN-Code

EAN bedeutet „Europäische Artikel-Numerierung". Wie der Name schon andeutet, sind damit nur die Ziffern 0...9 darstellbar. Jedes Zeichen besteht aus sieben Elementen; ein Element kann sowohl Strich als auch Lücke sein. (Auch die Lücken tragen hier also Information!) Daraus ergibt sich eine hohe Informationsdichte. Die Normung sieht maximal (je nach Größen-Version) acht oder 13 Ziffern vor. Die genauen Spezifikationen können bei der „Centrale für Coorganisation", Spicherstr. 55, 5000 Köln, angefordert werden.

13stellige EAN-Symbole *(Abb. 1.8.4.2)* bestehen aus 12 Nutzzeichen, davon sechs links und sechs rechts, abgesetzt durch ein Trennzeichen. Es sind nun unterschiedliche Zeichencodierungen definiert:

Zeichen	Satz A	Satz B	Satz C
0	0001101	0100111	1110010
1	0011001	0110011	1100110
2	0010011	0011011	1101100
3	0111101	0100001	1000010
4	0100011	0011101	1011100
5	0110001	0111001	1001110
6	0101111	0000101	1010000
7	0111011	0010001	1000100
8	0110111	0001001	1001000
9	0001011	0010111	1110100

Auf der linken Seite verwendet man Satz A und Satz B mit wechselnder Parität, auf der rechten Seite allein Satz C. Zwei schmale Striche dienen links und rechts als Randzeichen sowie als Trennzeichen in der Mitte.

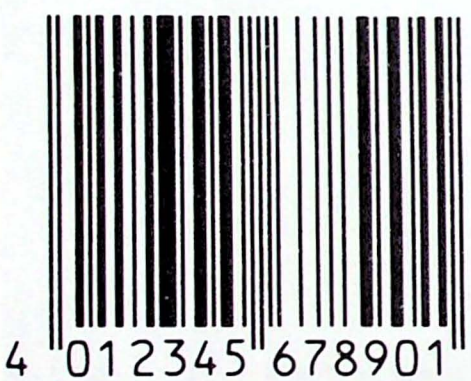


Abb. 1.8.4.2
EAN-Strichcode; er ist oft auf Verpackungen zu finden

Die Paritätsfolge auf der linken Symbolseite ermöglicht das Ermitteln einer Prüfziffer als 13. Stelle; die Zählung erfolgt dabei von rechts:

Zeichen	Paritätsfolge für linke Seite
0	A A A A A A
1	A A B A B B
2	A A B B A B
3	A A B B B A
4	A B A A B B
5	A B B A A B
6	A B B B A A
7	A B A B A B
8	A B A B B A
9	A B B A B A

A steht wieder für Zeichensatz A, B für Satz B.

Der EAN-Code ist heute „der" Standard für die Artikel-Numerierung im Handel. Sein Zweck ist leicht einsehbar: Schnellere Abwicklung an der Kasse, automatische Lagerbestands-Fortführung, selbständige Wiederbestellung bei Unterschreitung des Mindestbestands; schnelle Inventur z.B. mit tragbaren Computern und Lesestiften; Reduzierung von Falscheingaben.

Mit einem weit verbreiteten Irrtum sei schließlich noch aufgeräumt: Der EAN-Strichcode enthält nicht den Preis, auch nicht den Namen eines Produkts (er kann ja nur Ziffern darstellen). Vielmehr ist eine bestimmte Zifferngruppe einer bestimmten Produktgruppe zugeordnet.

Ein ähnlich aufgebauter, jedoch nicht kompatibler und bei uns auch kaum gebräuchlicher Code heißt UPC (Universal Product Code).

2-aus-5-Code mit fünf Strichen

Auch beim 2/5-Code lassen sich nur die Ziffern 0...9 darstellen. Dabei gibt es eine Version mit fünf und eine mit nur drei Strichen pro Ziffer. Zunächst zur 5-Strich-Version.

Hier ist in den Lücken keine Information. Jede Ziffer besteht aus zwei breiten und drei schmalen Strichen. Das Druckverhältnis „breit zu schmal" ist 3:1. Der Vorteil ist die gegenüber dem EAN-Code recht hohe Drucktoleranz von +/−25%. Das ermöglicht die Verwendung auch auf kritischem Untergrundmaterial und mit

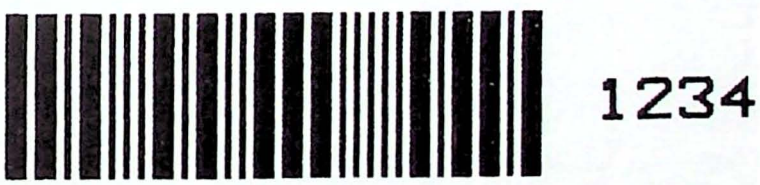


Abb. 1.8.4.3
2/5-Strichcode mit fünf Strichen je Ziffer

einfachen Druckverfahren. Der Nachteil ist die geringe Informationsdichte: Bei einer minimalen Strichbreite von 0,3 mm werden für eine Ziffer 6 mm benötigt. *Abb. 1.8.4.3* zeigt ein Druckbeispiel.

Wie beim EAN-Code wird mit einer Codetabelle gearbeitet. S1...S5 sind hier die fünf Striche pro Ziffer, eine 1 stellt einen breiten und eine 0 einen schmalen Strich dar. Die Start- und Stop-Zeichen bestehen aus nur drei Strichen:

Zeichen	S1	S2	S3	S4	S5
0	0	0	1	1	0
1	1	0	0	0	1
2	0	1	0	0	1
3	1	1	0	0	0
4	0	0	1	0	1
5	1	0	1	0	0
6	0	1	1	0	0
7	0	0	0	1	1
8	1	0	0	1	0
9	0	1	0	1	0
Start	1	1	0		
Stop	1	0	1		

Dieser Code ist wegen der großen Drucktoleranzen auch gut für die Herstellung mit Matrixdruckern geeignet.

2-aus-5-Code mit drei Strichen

Bei einer reduzierten Drucktoleranz von +/−10% bietet die 3-Strich-Version des 2/5-Codes eine deutlich höhere Informationsdichte: Bei 0,3 mm min. Strichbreite benötigt ein Zeichen (auch hier sind nur Ziffern darstellbar) 4,2 mm. Auch die Lücken zwischen den Strichen enthalten hier Information. Das Verhältnis breit zu schmal ist wieder 3:1. Jedes Zeichen besteht wieder aus fünf Elementen, nämlich drei Strichen (S1...S3) und zwei Lücken (L1, L2):

Zeichen	S1	L1	S2	L2	S3
0	0	0	1	1	0
1	1	0	0	0	1
2	0	1	0	0	1
3	1	1	0	0	0
4	0	0	1	0	1
5	1	0	1	0	0
6	0	1	1	0	0
7	0	0	0	1	1
8	1	0	0	1	0
9	0	1	0	1	0
Start	0	0	0		
Stop	1	0	0		

Abb. 1.8.4.4
2/5-Strichcode mit drei Strichen je Ziffer

0 bedeutet hier schmaler Strich bzw. schmale Lücke, 1 kennzeichnet einen breiten Strich bzw. eine breite Lücke. *Abb. 1.8.4.4* zeigt ein Beispiel für diesen von Datalogic eingeführten Code.

Programme als Strichcode

Zwei Mikrocomputer-Zeitschriften, Byte in den USA und mc in Deutschland, haben schon Strichcode zur Veröffentlichung von Programmen erprobt. mc hatte dazu sogar einen eigenen, sehr preiswerten Strichcode-Leser entwickelt *(Abb. 1.8.4.5)*.

Der mc-Strichcode ist folgendermaßen aufgebaut: Die Lücken enthalten keine Information. Breite Striche sind dreimal dicker als schmale. Ein breiter Strich stellt ein 1-Bit dar, ein schmaler ein 0-Bit. Jede Strichcode-Zeile beginnt mit zwei breiten Strichen zur Synchronisation (automatische Geschwindigkeits-Anpassung der

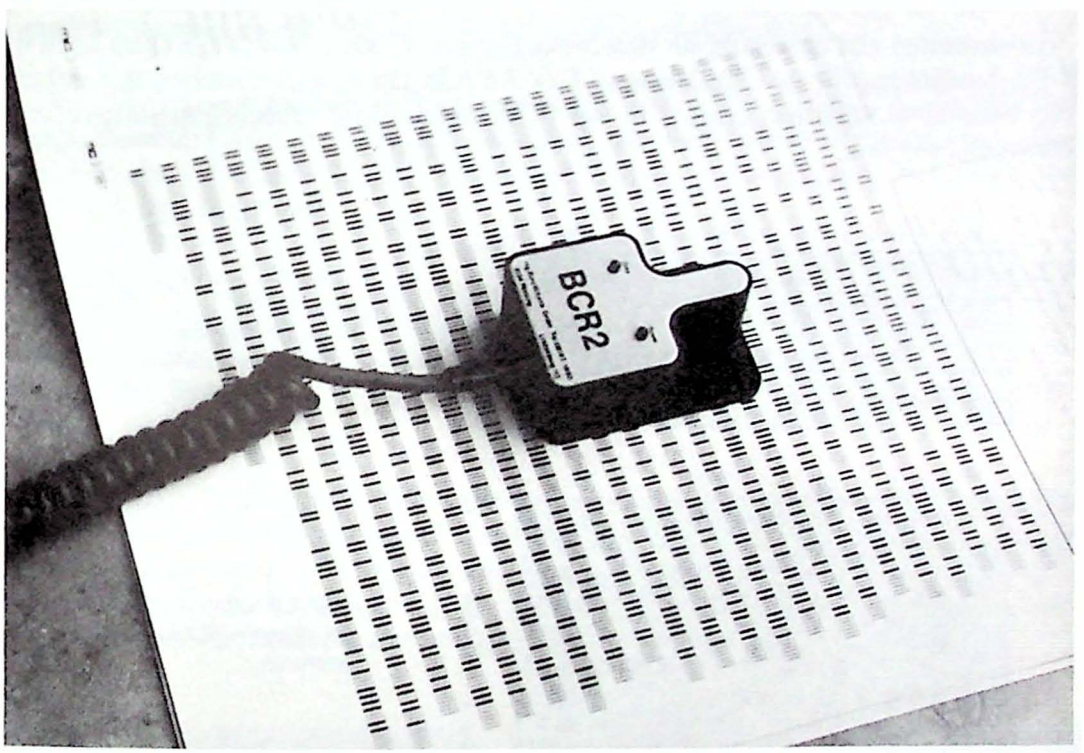


Abb. 1.8.4.5 Strichcode im mc-Format für Programm-Veröffentlichungen

Software), dann folgt ein Byte (acht Striche), das angibt, wie viele Datenbytes in dieser Zeile folgen. Nach den eigentlichen Datenbytes selbst folgen 16 Striche als 2-Byte-Prüfsumme über alle Datenbytes (Reihenfolge: höher-/niederwertiger Teil). Basic-Programme werden dabei nicht in dem (systemspezifischen) Intern-Code gedruckt, sondern als Folge von ASCII-Bytes.

Das Verfahren kommt allerdings nur für relativ kurze Programme in Betracht. Gegenüber einem normalen Programmlisting mit 2 mm Zeichenhöhe muß man für dasselbe Programm in Strichcode etwa den doppelten Platz rechnen. Sowohl Byte als auch mc haben daher den regelmäßigen Abdruck von Strichcode-Programmen eingestellt und teilweise durch andere Medien ersetzt, wie z.B. Floppy-Disk-Versand und Programmtransfer über das Telefonnetz (TEDAS = mc-Tele-Daten-Service, siehe Kapitel über Modems).

1.8.5 Lochstreifen-Datenformate

Obwohl der Lochstreifen heute als Speichermedium längst keine Rolle mehr spielt, hat er eine Reihe von Datenformaten geschaffen, die nach wie vor auch z.B. bei der Kassetten- oder Floppy-Aufzeichnung sowie bei der Datenfernübertragung eine Rolle spielen.

Lochstreifen gibt es sowohl für den 5-Bit-Baudot-Code *(Abb. 1.8.5.1)* als auch in 8-Bit-Ausführung für ASCII-Zeichen *(Abb. 1.8.5.2)*. Da ASCII-Zeichen nur sieben Bits benötigen, kann das achte z.B. zur Fehlererkennung mittels Paritätsprüfung verwendet werden.

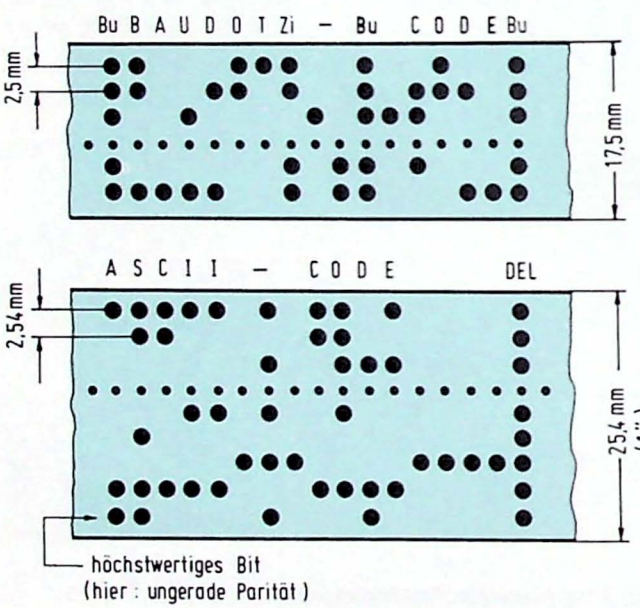


Abb. 1.8.5.1
Codierung eines Baudot-Lochstreifens

Abb. 1.8.5.2
So ist ein ASCII-Lochstreifen gestanzt

Nicht ganz in der Mitte des Lochstreifens (um ein falsches Einlegen in den Leser zu vermeiden) befindet sich eine Transportlochreihe mit etwas kleinerem Lochdurchmesser. Die übrigen Löcher codieren ein 1-Bit (ein 0-Bit entspricht keinem Loch an der entsprechenden Stelle).

Im Gegensatz zu reinem Text müssen für Speicherinhalte, wie bei jedem anderen externen Speichermedium auch, besondere Vereinbarungen getroffen werden. So muß festgelegt werden, in welcher Reihenfolge Anfangsadresse, Zahl der Datenbytes bzw. Endadresse sowie eine Prüfsumme aufgezeichnet bzw. gestanzt werden sollen. Erwähnt werden muß noch, daß alle hier erwähnten Formate aus der Blütezeit der 8-Bit-Prozessoren stammen. Da diese gewöhnlich nur 16 Adressenbus-Leitungen besitzen, sind auch nur 16 Adressenbits für die Aufzeichnung vorgesehen.

Das BNPF-Format

Böse Zungen behaupten, das BNPF-Format sei von der Papierindustrie erfunden worden, weil es eine vergleichsweise große Streifenlänge pro Byte benötigt. Kein Wunder: Jedes Bit wird als ein Baudot- oder ASCII-Zeichen gestanzt *(Abb. 1.8.5.3)*. Ursprünglich hatte Intel dieses Format als Datenträger zur Programmierung kundenspezifischer Festwertspeicher (ROMs) eingesetzt.

Der „Vorspann" enthält definitionsgemäß mindestens 25 Bu-Umschaltzeichen (beim Baudot-Code, zur Umschaltung auf die Buchstaben-Zeichenebene) oder DEL-Zeichen (bei ASCII). In beiden Codes sind das die Zeichen, bei denen alle Löcher ausgestanzt sind.

Der Beginn eines Bytes wird mit dem Zeichen B gekennzeichnet. Ein P (positiv) steht für ein 1-Bit, ein N (negativ) steht für ein 0-Bit. Nach acht Bits, d.h. am Byte-Ende, wird ein F (finish) gestanzt. Nach dem letzten Datenbyte folgt ein „Nachspann", der wiederum aus mindestens 25 Bu- bzw. DEL-Zeichen (je nach Code) besteht.

Der BNPF-Code übermittelt weder eine Anfangsadresse noch eine Prüfsumme. Eine Fehlererkennung ist einfach dadurch möglich, daß bei einem fehlerhaften Bit innerhalb eines Bytes vermutlich ein anderes Zeichen als P oder N herauskommt, was leicht erkannt werden kann. Diese Primitiv-Fehlererkennung wird allerdings mit der hohen Redundanz, also mit einer geringen Speicherdichte erkauft.

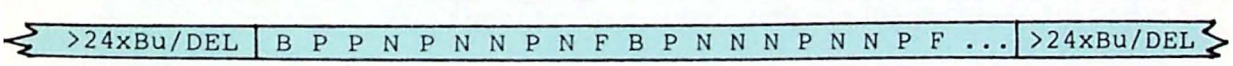


Abb. 1.8.5.3 Format des BNPF-Codes (bereichert nur Papier-Fabrikanten)

Das Motorola-S-Format

Mit wesentlich weniger Papier bzw. Speicherplatz kommt das von Motorola eingeführte S-Format aus. Es heißt so, weil jede Zeile mit dem Buchstaben S beginnt *(Abb. 1.8.5.4)*.

Die erste Zeile enthält die vierstellige Anfangsadresse (4 Ziffern = 2 Bytes) sowie einen Dateinamen (Filenamen), der aus ASCII-Zeichen besteht, die jedoch (wie alle anderen Bytes der Aufzeichnung) hexadezimal verschlüsselt sind. Ein „B" wird z.B. als „42" aufgezeichnet. Am Ende der ersten Zeile steht ein Prüfsummen-Byte.

In jeder Datenzeile steht nach dem S zunächst eine Hex-Ziffer für die Zeilenart; in der ersten Zeile (die den Filenamen enthält) ist sie Null. Dann folgen zwei Hex-Ziffern (ein Byte) als Zahl der Datenbytes dieser Zeile, die Datenbytes selbst und schließlich wieder ein Byte (zwei Ziffern) als Zeilen-Prüfsumme. Die letzte Zeile enthält keine Datenbytes mehr, sondern nur noch das Adressenfeld und ein komplementiertes Prüfsummen-Byte.

```
S008000044415441204 92F4FF3
S10800001122334455F8
S9030000FC
```

Abb. 1.8.5.4
Motorolas S-Format für Lochstreifen

Das 6502-Hex-Format

Als MOS Technology den Prozessor 6502 entwickelt hatte, führte diese Firma gleichzeitig mit ihm ein Format ein, das es gestattete, innerhalb einer Hex-Datei mehrere unterschiedliche Speicherbereiche zu übertragen. Zu diesem Zweck erhielt nun jede Zeile vier Ziffern für die Adresse. Ferner wurde die Fehlererkennung durch Verlängern der Prüfsumme auf 16 Bits (2 Bytes, 4 Ziffern) verbessert. Wie *Abb. 1.8.5.5* zeigt, beginnt bei diesem Format jede Zeile mit einem Strichpunkt. Wie beim Motorola-S-Format wird ein Byte als Folge von zwei ASCII-Zeichen (Hex-Ziffern) übertragen. Ein Filename ist hier allerdings nicht vorgesehen.

Nach dem Strichpunkt stehen in jeder Zeile die Zahl der Datenbytes (dezimal max. 24 = hex 18), die vierstellige Anfangsadresse für diese Zeile, die Datenbytes selbst und eine 2-Byte-Prüfsumme. Adresse und Prüfsumme werden in der Reihenfolge höher-/niederwertiges Byte übermittelt. In der letzten Zeile steht als Datenbyte-Anzahl 00 und statt einer Adresse die Gesamtzahl der übertragenen Zeilen (inklusive der letzten) als 2-Byte-Wert. Die nun noch folgende Prüfsumme ist hier natürlich mit der Zeilenzahl identisch.

```
;1808004C9C08201809205E09105AC9BBD0F720AD0984FA84FB207E09FE
;18081809B044F05E85F4207E09B03B85FD207E09B03485FCA5FDA20C60
;10083006DDB509B005DDB409B006CACA10F23007B4
;0000040004
```

Abb. 1.8.5.5 Das MOS-Technology-6502-Format

```
:100000007A2F577B2F5F132100003E11E519D21282
:1000100000E3E1F579174F7817477D176F7C176775
:0D002000F13DC20C00B77C1F577D1F5FC96A
:00000001
```

Abb. 1.8.5.6 Aufzeichnung im Intel-Hex-Format

Das Intel-Hex-Format

Auch für Intel war das BNPF-Format auf Dauer zu papierintensiv; als Folge davon entstand das Intel-Hex-Format, das leider mit den beiden anderen nicht kompatibel ist *(Abb. 1.8.5.6)*.

Nach einem Doppelpunkt steht auch hier die Anzahl der Datenbytes der Zeile und dann die vierstellige Anfangsadresse. Dann jedoch folgen zwei Ziffern für die Kennzeichnung des Record-Typs (Art der Aufzeichnung). Bei Datenzeilen ist dieser Record-Typ 00. Die letzte Zeile enthält keine Daten und besitzt den Record-Typ 01. Ähnlich wie beim Motorola-S-Format steht am Schluß jeder Zeile ein einziges Prüfsummen-Byte.

1.9 Schaltungstechnik bei Mikrocomputern

1.9.1 Allgemeines

Die tatsächliche Ausführung der Schaltung eines Mikrocomputers hängt natürlich sehr stark von den verwendeten Bausteintypen ab. Grundsätzlich besitzt aber jede die in *Abb. 1.9.1.1* dargestellte Struktur.

Der Mikroprozessor (CPU) erhält den nötigen Takt von einem Taktoszillator; in manchen CPUs ist er allerdings schon (bis auf den Quarz) integriert. Normalerweise wird ein symmetrisches Rechtecksignal als Takt gefordert.

Um den RAM-, ROM- und I/O-Bausteinen des Systems bestimmte Adressenbereiche zuzuweisen, damit sich ihre Speicherzellen nicht überschneiden, ist ein Adressendecoder erforderlich. Er generiert die sogenannten Chip-Select-Signale (Baustein-Auswahl) aus den höherwertigen Adressenleitungen. So kann z.B. ein 16-aus-4-Decoder aus den vier Leitungen A 12...15 sechzehn einzelne Bereiche von je 4 KByte decodieren.

In welcher Reihenfolge RAM, ROM oder I/O im Adressenbereich angeordnet sind, hängt vom jeweiligen CPU-Typ ab. So muß bei den CPUs 8080 und Z80 ab Adresse 0000 ROM bzw. EPROM vorhanden sein, da dort nach einem Reset mit der Programm-Abarbeitung begonnen wird *(Abb. 1.9.1.2)*. Der I/O-Bereich muß

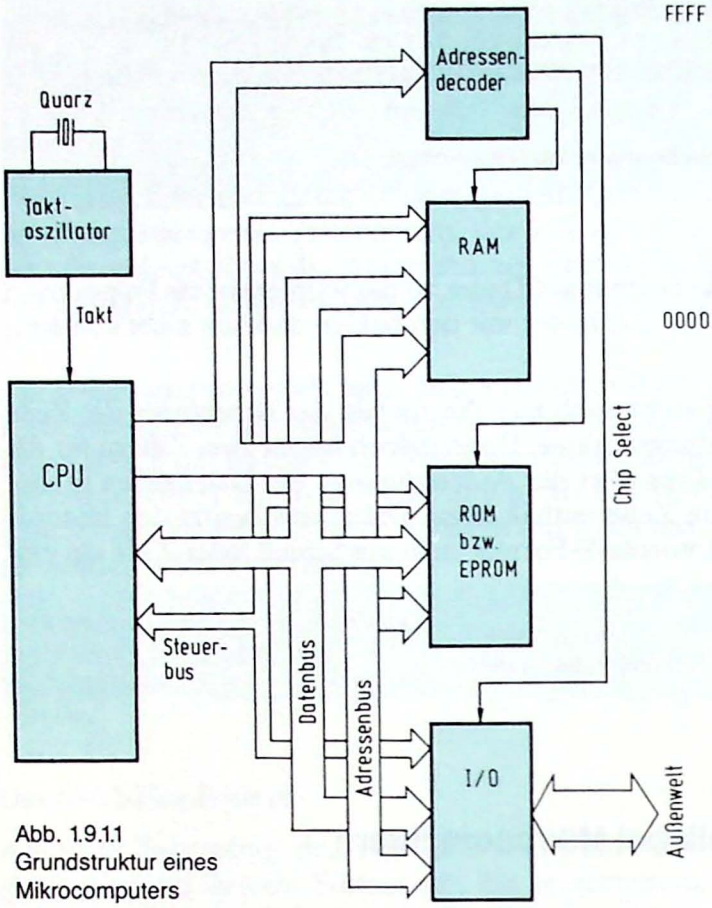


Abb. 1.9.1.1
Grundstruktur eines Mikrocomputers

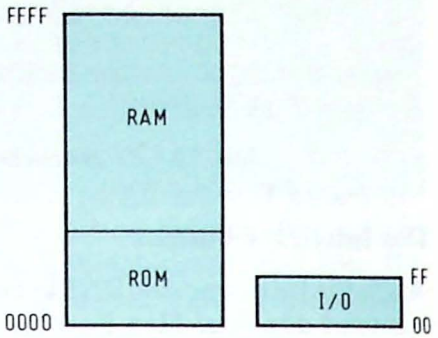


Abb. 1.9.1.2 Bei 8080, Z80 und anderen CPU-Typen unterscheidet man zwischen Speicher- und I/O-Adressen

nicht innerhalb des Speicher-Adressenbereichs liegen, da für seine Anwahl eine eigene CPU-Steuerleitung namens I/O-Request vorhanden ist. Allerdings sind damit nur die speziellen I/O-Befehle (IN, OUT usw.) auf Ein-/Ausgabe-Bausteine anwendbar.

Bei den Prozessoren 6800, 6809, 6502 und anderen muß dagegen am oberen Speicherende ROM oder EPROM liegen: Diese CPUs holen bei einem Reset die Startadresse des abzuarbeitenden Programms aus Speicherzellen zwischen FFF0 und FFFF (je nach Typ). Da es hier keine CPU-Steuerleitung für die Anwahl von I/O-Bausteinen gibt, müssen diese innerhalb des normalen Adressenbereichs liegen. Das RAM sollte bei den CPUs 6800 und 6502 bei 0000 beginnen, um für die Zero-Page-Adressierungsart dieser Prozessoren zur Verfügung zu stehen. Hieraus ergibt sich eine Adressenbelegung wie in *Abb. 1.9.1.3.* Auf Ein-/Ausgabe-Bausteine sind bei dieser Memory-Mapped-Anordnung der I/O-Chips alle CPU-Befehle uneingeschränkt anwendbar.

8-Bit-Prozessoren besitzen gewöhnlich 16 Adressenleitungen und können daher nur 64 KByte adressieren. Will man mehr Speicher implementieren, so geht das nur mit einem Trick. *Abb. 1.9.1.4* zeigt ein Beispiel (CBM-8096, CPU=6502):

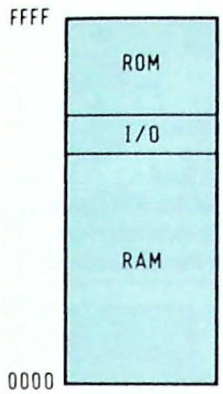


Abb. 1.9.1.3
6800, 6502 und einige andere Prozessoren sprechen I/O-Bausteine „memory-mapped" an

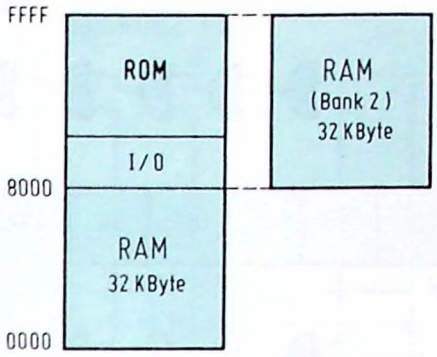


Abb. 1.9.1.4
64 KByte RAM werden bei 16 Adressenbits erst durch eine Bank-Umschaltung möglich

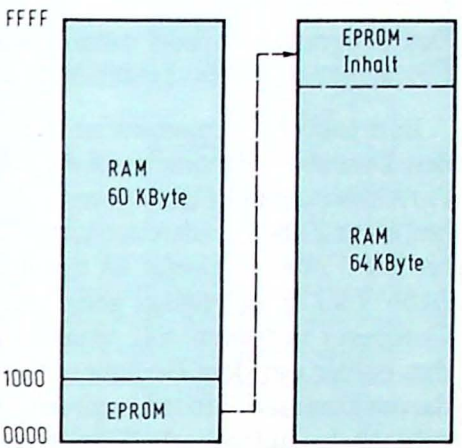


Abb. 1.9.1.5
Eine Bankumschaltung ist auch bei den meisten CP/M-Rechnern eingebaut. Beim Einschalten ist noch das EPROM eingeblendet, danach stehen aber 64 KByte RAM zur Verfügung

Im gleichen Adressenbereich wie ROM und I/O liegt hier eine zweite RAM-Speicherbank mit nochmals 32 KByte. Die Umschaltung erfolgt z.B. mit einer I/O-Port-Leitung. Natürlich ist das ROM abgeschaltet, wenn die zweite RAM-Bank eingeschaltet wird. In die erste RAM-Bank muß daher vorher ein kleines Hilfsprogramm geschrieben werden, das den Datentransfer von und zur zweiten Bank abwickelt. Rein theoretisch könnte man natürlich die zweite RAM-Bank auch in denselben Adressenbereich wie die erste legen; das hätte aber den Haken, daß ein Transfer von Daten zwischen beiden RAM-Bänken schwieriger abzuwickeln wäre.

Eine besondere Rolle spielt das „Banking" beim Floppy-Disk-Betriebssystem CP/M, worauf später noch eingegangen wird. Es arbeitet mit den CPU-Typen Z80 und 8080. Beim Einschalten ist z.B. in die untersten 4 KByte (Adressen 0000...0FFF) ein EPROM eingeblendet, das lediglich ein Hilfsprogramm zum Laden des Betriebssystems von der Floppy enthält *(Abb. 1.9.1.5)*. Da dieses Betriebssystem aber am unteren Speicherende RAM voraussetzt, muß sich das EPROM nach dem Laden von CP/M selbst abschalten. Da man sich bekanntlich nicht an den eigenen Haaren aus dem Sumpf ziehen kann, geht das nur mit einem Trick: Das EPROM kopiert sich zunächst selbst in den Bereich hex F000...FFFF.

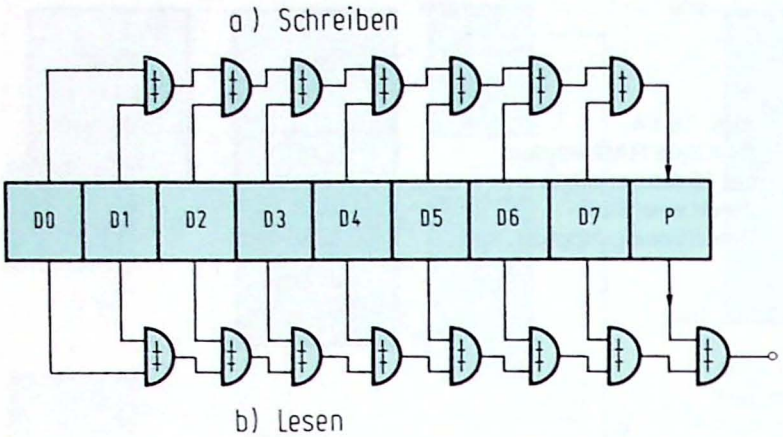


Abb. 1.9.1.6
Prinzip der Erkennung von Speicherfehlern durch Paritätsprüfung

Das Programm springt dann dorthin und schaltet das EPROM ab. Nun kann CP/M unterhalb F000 geladen und gestartet werden.

In manchen Computern ist der Speicher nicht 8 oder 16 Bit „breit", wie es für den Datenbus eigentlich erforderlich wäre, sondern wird mit einem zusätzlichen Paritätsbit ergänzt. Eine geeignete Schaltung stellt fest, ob die Zahl der 1-Datenbits einer Zelle gerade oder ungerade ist und setzt danach das zugehörige Paritätsbit auf 0 oder 1. Damit ist es möglich, Einzelbitfehler im Speicher zu erkennen *(Abb. 1.9.1.6)*. Allerdings gehen die Meinungen darüber auseinander, wie sich der Computer verhalten soll, wenn ein solcher Paritätsfehler erkannt wird. Wenn er den Fehler nur dem Bediener meldet, so besteht die Gefahr, daß dieser sich nicht darum kümmert und u.U. mit falschen Daten weiterrechnet (wenn es um Finanzen geht, ist das untragbar!). Schaltet der Computer dagegen zur Sicherheit ganz ab, so hat das auch dann den Verlust aller vorher eingegebenen Daten zur Folge, wenn der Fehler in einem nicht relevanten Bereich oder z.B. in einem Textkommentar auftrat.

Insbesondere bei unbedienten Einplatinen-Computern, z.B. in automatisierten Steuer- und Regelungseinrichtungen, finden sich schließlich noch sogenannte Watchdog-Schaltungen (Wachhund). Wie die Parity-Prüfung dienen sie ebenfalls dem Erkennen von Fehlern. Allerdings wird hier nur der Extremfall „Programm

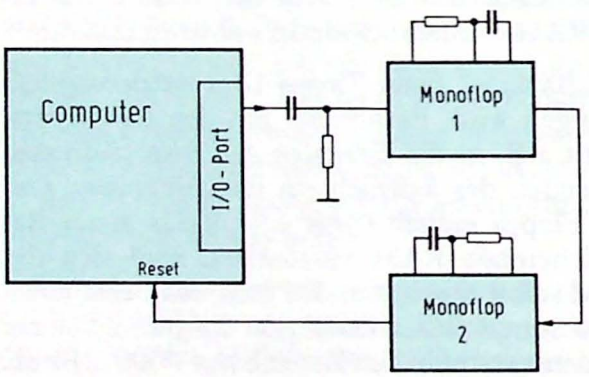


Abb. 1.9.1.7
Ein „Aufhängen" des Rechners z.B. durch Störimpulse kann man mit einer Watchdog-Schaltung erkennen und einen Reset veranlassen

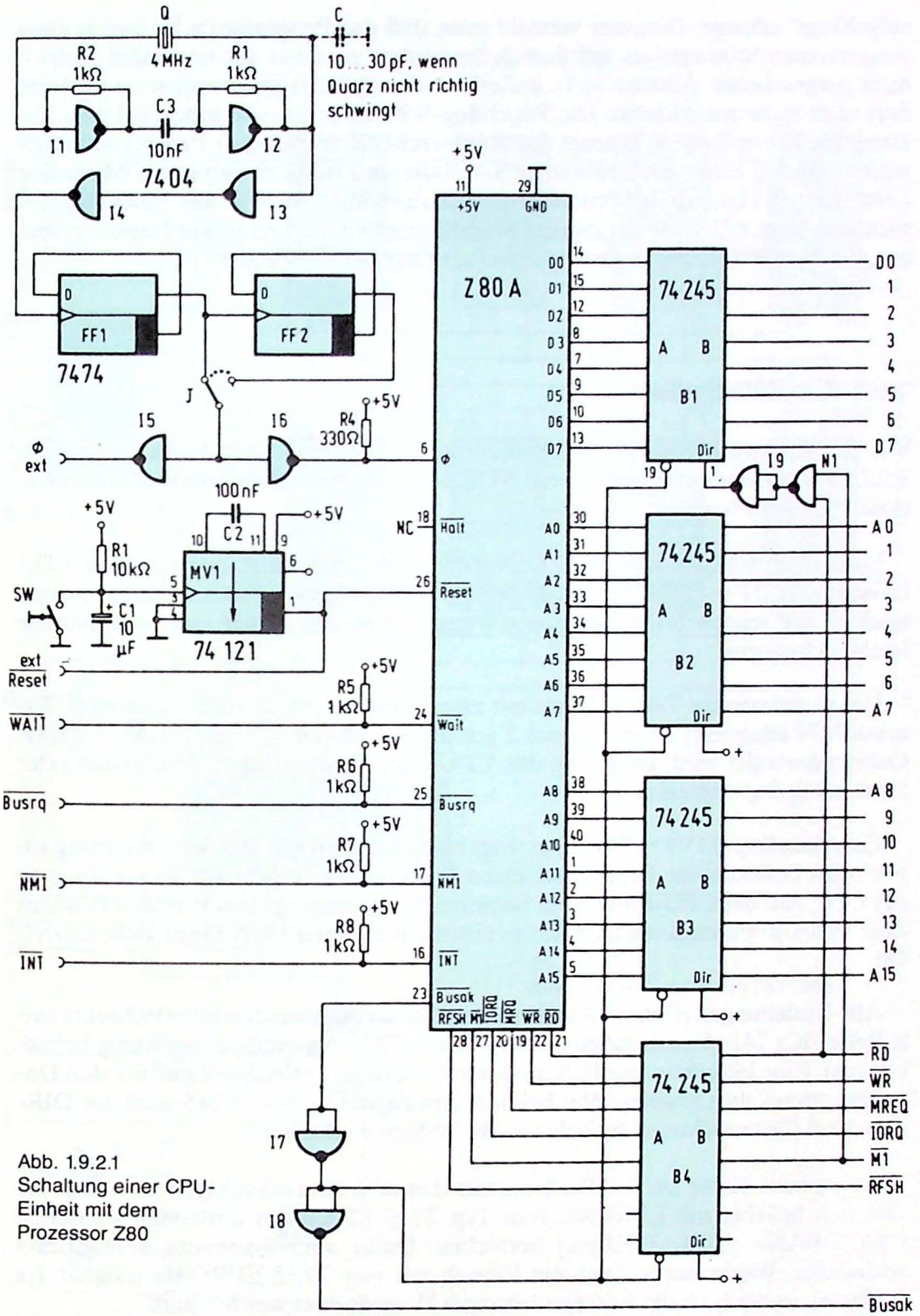


Abb. 1.9.2.1
Schaltung einer CPU-Einheit mit dem Prozessor Z80

aufgehängt" erkannt: Darunter versteht man, daß das Programm z.B. wegen eines aufgetretenen Störimpulses auf dem Adressenbus zu einer im normalen Betrieb nicht vorgesehenen Adresse (u.U. außerhalb des ROMs) gesprungen ist und von dort nicht mehr zurückkehrt. Die Watchdog-Schaltung erkennt einen solchen Zustand: Im Normalbetrieb erzeugt das (entsprechend ausgelegte) Programm regelmäßig Impulse an einem bestimmten I/O-Port und triggert damit ein Monoflop *(Abb. 1.9.1.7)*. Hat sich der Prozessor nun „aufgehängt", so fällt das Monoflop zurück und kann z.B. über ein zweites Monoflop einen kurzen Reset-Impuls erzeugen, um die CPU wieder in einen definierten Zustand zu bringen.

1.9.2 Ein Z80-System

Wie das Kernstück eines Z80-Mikrocomputers aufgebaut sein kann, zeigt *Abb. 1.9.2.1.* Dieser Computer wurde vom NDR 1985 in seiner Sendereihe „Mikroelektronik" als Selbstbaugerät propagiert.

Links oben befindet sich ein 4- oder 8-MHz-Taktgenerator, der aus zwei TTL-Invertern (I1, I2 = ⅙ 7404) gebildet wird. Zwei nachgeschaltete weitere Inverter machen aus seinem noch eher sinusförmigen Ausgangssignal eine steilflankige Rechteckfrequenz.

Der so gewonnene Takt wird nun mit zwei Flipflops (beide sind in einem IC namens 7474 integriert) zweimal durch 2 geteilt. Je nachdem, ob ein 4- oder 8-MHz-Quarz verwendet wird, kann man den CPU-Takt (2 MHz) nach dem ersten oder zweiten Flipflop abgreifen.

Ein Monoflop (MV1 = 74121) erzeugt beim Anlegen der Betriebsspannung sowie beim Drücken der Reset-Taste einen kurzen Low-Impuls am Reset-Eingang der CPU. Auf der CPU-Karte nicht benutzte Steuereingänge des Prozessors liegen über Pull-Up-Widerstände an 5 V, um einen eindeutigen High-Pegel sicherzustellen.

Alle Busleitungen (Daten-, Adressen- und Steuerbus) werden über bidirektionale Puffer-ICs 74245 nach außen geführt, da die CPU-Anschlüsse nur wenig belastbar sind. Eine bidirektionale Pufferung wäre allerdings prinzipiell nur für den Datenbus erforderlich gewesen. Nur bei dem ihm zugeordneten 74245 wird der DIR-Anschluß (Datenrichtung) deshalb vom R/W-Signal gesteuert.

Eine zweite Karte *(Abb. 1.9.2.2)* enthält den zum Betrieb nötigen Speicher. Sie läßt sich beliebig mit EPROMs vom Typ 2716 (2 KByte) und/oder statischen CMOS-RAMs (6116, 2 KByte) bestücken. Dafür sind insgesamt acht Sockel vorhanden. Wahlweise ist auch ein Betrieb mit vier 2732-EPROMs möglich (je 4 KByte), wobei auch die Adressenleitung A11 verdrahtet werden muß.

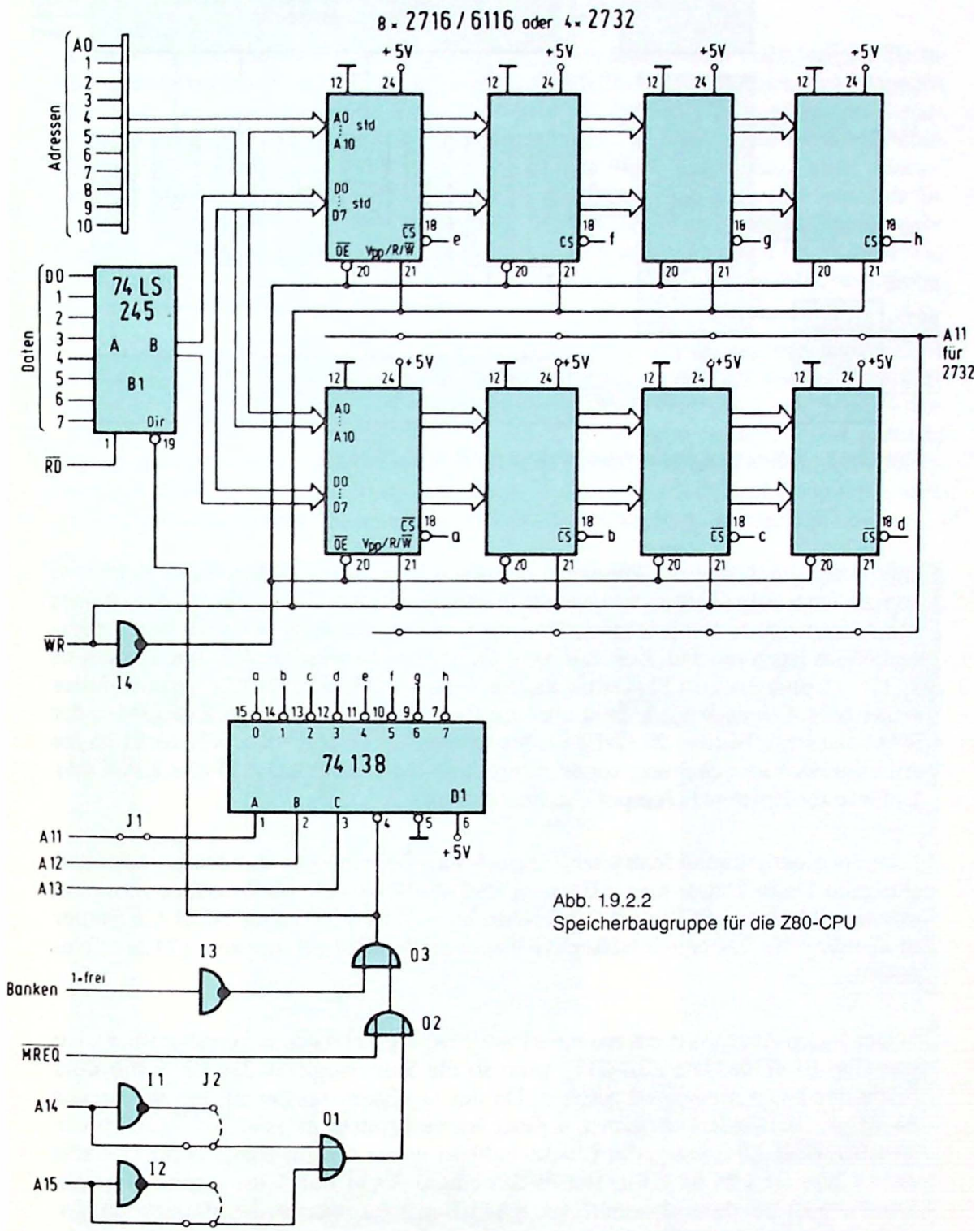


Abb. 1.9.2.2
Speicherbaugruppe für die Z80-CPU

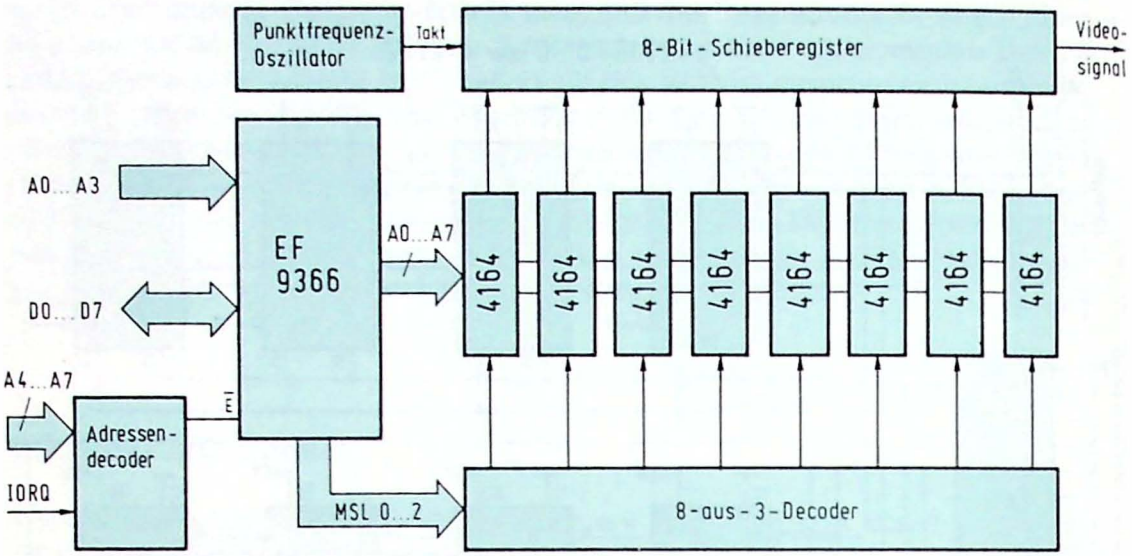


Abb. 1.9.2.3 Aufbau eines Grafik-Video-Interface mit dem IC EF 9366

Je nach Brückenlegung können der Speicherkarte unterschiedliche Adressen zugewiesen werden. Ferner ist über den Eingang „Banken" ein völliges Abschalten der Karte möglich, wenn man eine zweite Speicher-Bank in den gleichen Adressenbereich legen möchte. Zur Adressen-Decodierung werden die fünf Leitungen A 11...15 über einige TTL-Gatter an den 8-aus-3-Decoder 74138 geführt. Jeder seiner acht Ausgänge a...h dient als Chip-Select-Signal für einen EPROM- oder RAM-Baustein. Werden 2732-EPROMs verwendet, so legt man A11 nicht an die Adressendecoder-Schaltung, sondern direkt an den Anschluß A11 des EPROMs. Dafür ist die Brücke J1 (Jumper 1) aufzutrennen.

Dieser Konfiguration fehlt jetzt nur noch eine Möglichkeit zur Daten-Ein- und -Ausgabe. Dafür könnte man z.B. einen Z80-PIO-Baustein als Interface zu einem externen Terminal mit Tastatur und Bildschirm einsetzen. Beim NDR-Computer ist allerdings als Tastatur-Interface ein gewöhnlicher Puffer-Baustein (74245) eingesetzt.

Der Video-Ausgabe dient ein direkt am Bus angeschlossener Grafik-Prozessor vom Typ EF 9366. Die Z80-CPU kann so die Steuerregister des EF 9366 über bestimmte I/O-Adressen ansprechen. Da der Grafikprozessor mit einem eigenen 64-KByte-Bildwiederholspeicher ergänzt ist, verbraucht er keinen CPU-Arbeitsspeicher. *Abb. 1.9.2.3* zeigt das Blockschaltbild dieser Grafik-Baugruppe: Die acht 4164-Chips (je 1 × 64 KBit) stellen das Video-RAM dar. Eine eigene Refresh-Schaltung ist für diese dynamischen RAM-Bausteine wegen des dauernden Zugriffs durch den Grafik-Prozessor nicht erforderlich.

1.9.3 Ein 6504-Einplatinen-Computer

Das im letzten Kapitel besprochene Z80-System ergibt einen vollständigen Tischcomputer mit allem Komfort. Ein ganz anderes Anwendungsgebiet deckt dagegen der hier vorgestellte Einplatinen-Computer ab. Er dient in intelligenten Steuerungen als Ersatz für sonst nötige umfangreiche Logikschaltungen herkömmlicher Technik. Beispielsweise kann er Herzstück einer Heizungsregelung, einer Alarmanlage, eines Strichcode-Interface oder eines Verschlüsselungsgeräts sein. Der für solche Aufgaben nötige Speicherplatz ist recht gering. Entsprechend einfach sieht die Schaltung aus *(Abb. 1.9.3.1)*. Dieser Computer wurde übrigens schon 1981 von der Fachzeitschrift mc als „EMUF" veröffentlicht (Einplatinen-Mikrocomputer für universelle Festprogrammierung).

Die verwendete CPU 6504 besitzt den gleichen Befehlssatz wie ihre größere Schwester 6502. Beide unterscheiden sich lediglich in der Zahl der herausgeführten Adressenbus-Leitungen. Ferner besitzt der Prozessor 6504 keinen NMI-Anschluß, der aber meist verzichtbar ist. Der EMUF selbst ist nicht dafür gedacht, Programme mit ihm zu entwickeln. Vielmehr erstellt man das Programm für die jeweilige Anwendung zunächst auf einem beliebigen 6502-Tischcomputer *(Abb. 1.9.3.2)*. Dann „brennt" man es in ein EPROM und steckt es auf den EMUF.

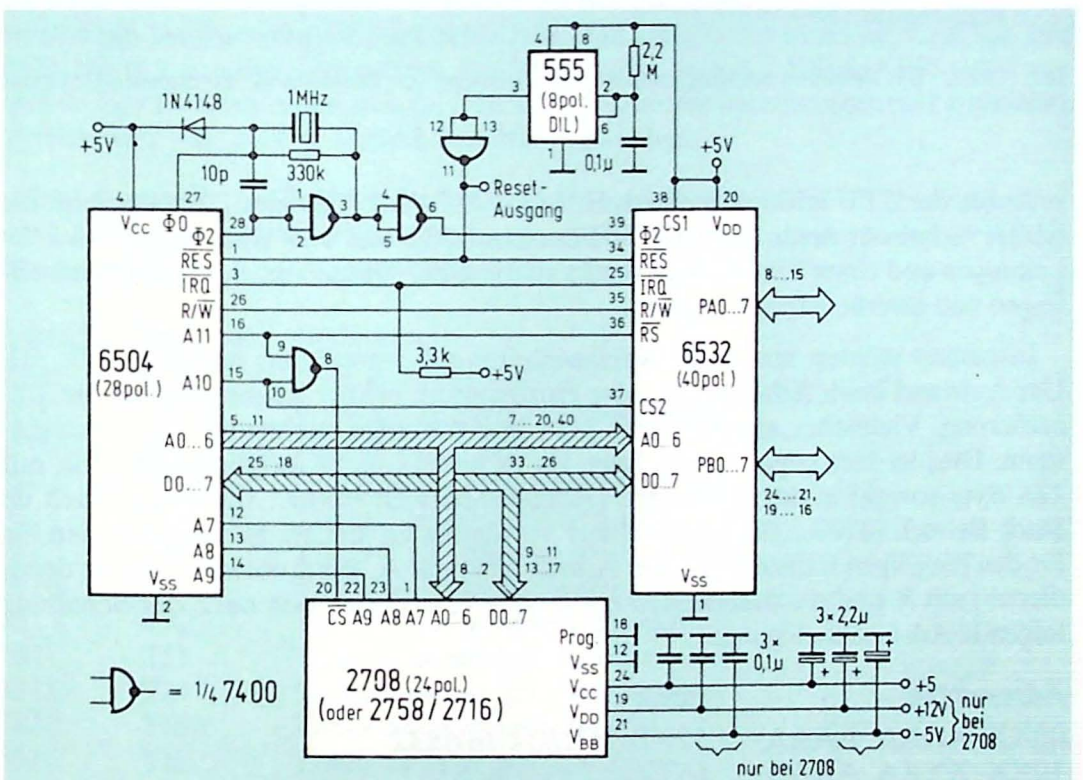


Abb. 1.9.3.1 Schaltung eines kleinen Einplatinen-Computers mit 1 oder 2 KByte EPROM, 128 Byte RAM, 16 I/O-Leitungen und einem Timer. Als CPU dient die 6502-kompatible Version 6504

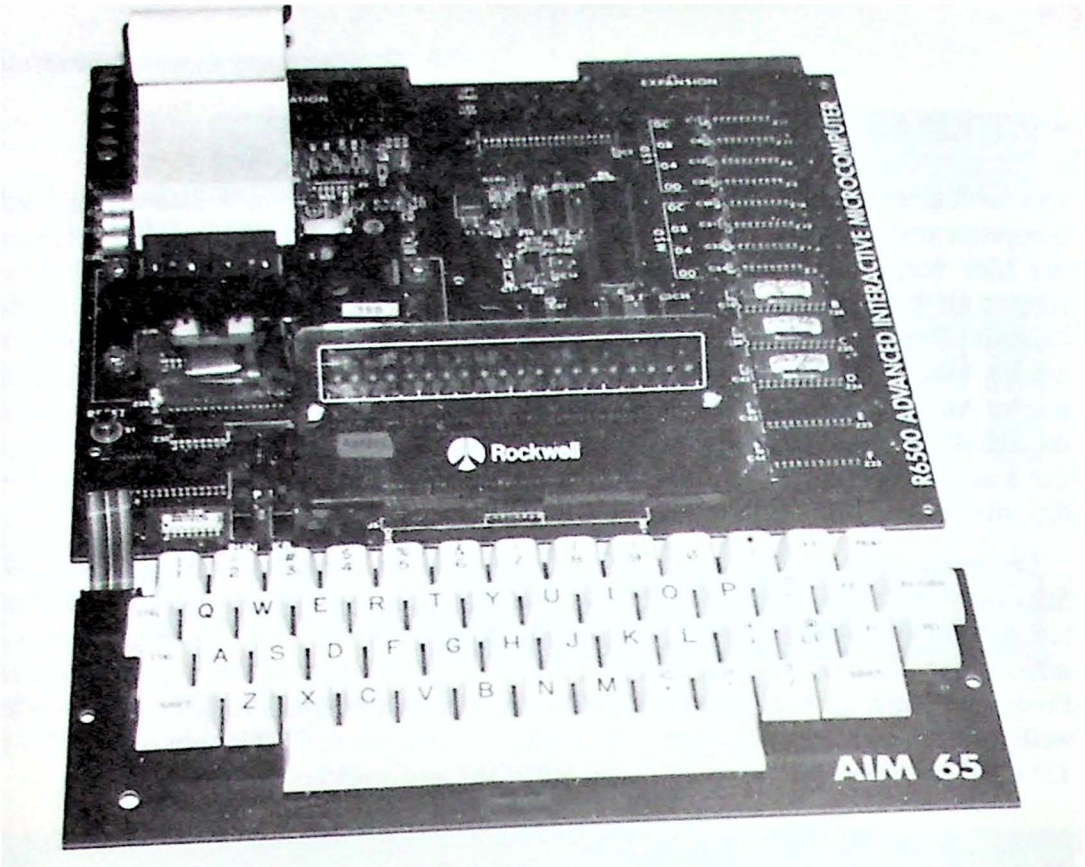


Abb. 1.9.3.2 Ein kleines Entwicklungssystem zur Erstellung von Software für Einplatinen-Computer (AIM-65)

Außer der CPU 6504 und einem EPROM (2708, 2758 oder 2716) enthält die EMUF-Schaltung noch einen RIOT-Baustein 6532 mit 128 Byte RAM, 16 I/O-Leitungen und einen Timer. Über ihn kann die CPU Daten von der Außenwelt abfragen und externe Abläufe steuern.

Insgesamt werden nur zwölf Adressenleitungen verwendet, nämlich A 0...11. Um Aufwand beim Adressendecoder einzusparen, erfolgt keine vollständige Decodierung. Vielmehr „spiegelt" sich jede Speicherzelle mehrmals im Adressenraum. Dies ist hier sogar beabsichtigt, da es anders nicht möglich wäre, mit nur 128 Byte sowohl in der Zero Page (Adressenbereich 0000...00FF) als auch im Stack-Bereich (0100...01FF) RAM zur Verfügung zu stellen. Bezeichnet man ein für den jeweiligen Baustein gültiges Adressenbit mit A, ein ignoriertes (nicht decodiertes) mit X und die decodierten mit 0 oder 1, so ergibt sich nach der Schaltung folgende Adressenbelegung:

Adressenbits	Inhalt
00XX XAAA AAAA	128 Byte RAM im 6532
10XX XXXA AAAA	I/O und Timer im 6532
11AA AAAA AAAA	EPROM (1 KByte)
01AA AAAA AAAA	nicht belegt

Hierbei wurden nicht 16, sondern nur 12 Adressenbits aufgeführt, da die höherwertigen vier (A 12...15) nicht decodiert sind. Das bedeutet, daß die Hex-Adresse FFFF gleichwertig ist mit 0FFF. Da beim 6502 bzw. 6504 der Reset-Vektor bei FFFC/FFFD liegt, wird er auch dann richtig geholt, wenn er im EPROM bei 0FFC/0FFD steht. Der Vorteil ist, daß im Entwicklungssystem der komplette EPROM-Inhalt im Bereich 0C00...0FFF programmiert werden kann, in dem normalerweise in Tischcomputern auch RAM zur Verfügung steht.

Schlüsselt man die Adressenbelegung hexadezimal auf, so ergibt sich folgendes Bild:

RAM	X000...X07F; X080...X0FF; X100...X17F; X180...X1FF usw. bis X3FF
EPROM	XC00...XFFF
I/O, Timer	X800...X81F (32mal dupliziert bis XBFF)
Frei	X400...X7FF

Hier wird auch deutlich, daß tatsächlich sowohl im Zero-Page- als auch im Stackbereich RAM zur Verfügung steht. Allerdings ist die Stack-Zelle 01FF identisch mit der Zero-Page-Zelle 007F und auch mit 00FF. Bei einer maximalen Stacktiefe von z.B. 16 Byte (acht Unterprogramm-Ebenen) stehen aber immerhin noch die 112 Zero-Page-Zellen 0000...006F zur Verfügung, was für die meisten Steuerungs-Anwendungen ausreicht. Aus dem gleichen Grund ist auch für das EPROM nur 1 KByte decodiert. Bei Verwendung eines 2-KByte-EPROMs kann man in ihm zwei unterschiedliche Betriebsprogramme unterbringen und am Pin 19 (A10 beim 2716) z.B. mit einem Schalter auswählen.

Die Auswahl von RAM oder I/O- und Timer-Registern im 6532 erfolgt über seinen Eingang RS- (RAM Select). Ist er auf High-Pegel, so ist der I/O- und Timer-Bereich aktiviert. Im EMUF ergeben sich folgende Adressen:

Adresse	Kurzbez.	Bedeutung
0800	PA	Port-A-Datenregister
0801	PAD	Port-A-Datenrichtungs-Register
0802	PB	Port-B-Datenregister
0803	PBD	Port-B-Datenrichtungs-Register
0814	T1	Timer 1 µs ohne Interrupt
0815	T8	Timer 8 µs ohne Interrupt
0816	T64	Timer 64 µs ohne Interrupt
0817	TK	Timer 1024 µs ohne Interrupt
081C	TI1	Timer 1 µs mit Interrupt
081D	TI4	Timer 4 µs mit Interrupt
081E	TI64	Timer 64 µs mit Interrupt
081F	TIK	Timer 1024 µs mit Interrupt

(Ein Taktzyklus entspricht bei 1 MHz genau einer Mikrosekunde. Näheres über das IC 6532 findet sich im Kapitel über Peripherie-Bausteine.)

2 Peripheriegeräte

Unter „Peripherie" versteht man grundsätzlich alles, was nicht im Computer selbst, sondern in einem eigenen Gehäuse untergebracht ist. Die Verbindung erfolgt nicht über den internen Daten- und Adressenbus, sondern über Schnittstellen wie Centronics (parallel), RS-232C alias V.24 (seriell) oder herstellerspezifische Anschlüsse. Dieses Kapitel unterscheidet drei Arten von Peripheriegeräten — solche zur Ausgabe, zur Kommunikation (Ein- und Ausgabe) und nur zur Eingabe.

2.1 Ausgabegeräte

2.1.1 Matrix-Drucker

Drucker dienen der Ausgabe von Schriftzeichen oder Grafiken auf eine Unterlage — vorzugsweise auf Papier. Setzt ein Drucker die Schriftzeichen aus einzelnen Punkten zusammen, wie das schon im Zusammenhang mit Video-Controllern besprochen wurde, so spricht man von Matrix-Druckern. Und wie bei der Video-Ausgabe ist die erzielbare Druckqualität umso besser, je mehr Punkte für ein Zeichen zur Verfügung stehen. Technisch gibt es sehr unterschiedliche Verfahren, dunkle Punkte auf eine helle Unterlage zu drucken.

Elektrosensitiv-Drucker

Sehr preiswert lassen sich Drucker herstellen, die mit schwarzem Papier arbeiten, das jedoch oberflächlich mit einer sehr dünnen, hellen Aluminiumschicht bedampft ist. Läßt man über zwei dicht nebeneinanderliegende Kontakte, die die Alu-Schicht berühren, genügend viel Strom fließen (etwa 1 A), so wird aus dieser ein kleiner Punkt herausgebrannt, und das darunterliegende, schwarz gefärbte Papier wird sichtbar. Es entsteht so ein dunkler Punkt auf hellem Grund.

Ordnet man nun z.B. sieben solche Kontaktpaare übereinander an und bewegt sie horizontal über das Alu-Papier, so kann man eine ganze Zeile von Schriftzeichen drucken, indem man nur zu den richtigen Zeitpunkten den Strom für das jeweilige Kontaktpaar ein- und ausschaltet. Dann bewegt man das Papier vertikal um einen Zeilenabstand weiter und druckt die nächste Zeile. Die erzielbare Druckgeschwindigkeit beträgt etwa eine bis drei Zeilen pro Sekunde.

Zwar sind solche Drucker einfach und billig herstellbar, das aluminiumbeschichtete Papier jedoch ist recht teuer, schwer erhältlich und sehr empfindlich gegen Fingerabdrücke und Knittern. Elektrosensitive Drucker sind daher nur noch selten zu finden.

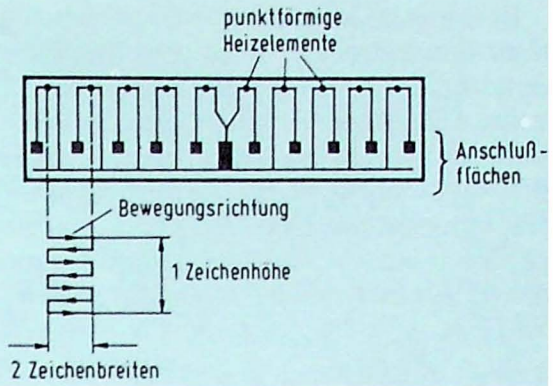


Abb. 2.1.1.1
Anordnung der Heizelemente bei einem Thermodrucker

Thermodrucker

Ähnliche Vor- und Nachteile, wie sie bei den elektrosensitiven Druckern genannt wurden, gelten prinzipiell auch für Thermodrucker — der Drucker selbst ist preiswert, das nötige Spezialpapier dagegen oft schwer zu beschaffen und teuer. Immerhin ist aber Thermopapier optisch dem Normalpapier wesentlich ähnlicher und daher angenehmer zu lesen als das silbrig glänzende Alu-Papier.

Abb. 2.1.1.1 zeigt den Druckkopf eines Thermodruckers für 20 Zeichen Zeilenbreite. Er enthält auf einem flachen Keramik-Träger zehn punktförmige Heizelemente, die von der Drucker-Elektronik einzeln kurzzeitig erhitzt werden können. Das Thermopapier ist mit einer hitzeempfindlichen chemischen Schicht versehen, die bei Temperaturen oberhalb etwa 150 Grad Celsius dunkel wird, z.B. blau oder schwarz. Die dazu erforderliche elektrische Leistung ist im Vergleich zu elektrosensitiven und auch zu den weiter unten besprochenen Nadeldruckern recht gering, ebenso das Druckgeräusch.

Zum Schreiben einer Zeile wird der Kopf wechselweise nach rechts und links und gleichzeitig das Thermopapier langsam vorwärts bewegt. Jedes Heizelement schreibt bei einer Rechts- oder Linksbewegung genau eine horizontale Punktreihe zweier nebeneinanderliegender Zeichen. Wenn für jedes Zeichen für eine 5×7-Matrix sieben Punktzeilen plus eine als Abstand zur nächsten Schriftzeile geschrieben werden, ergeben sich pro Zeile acht Hin- und Herbewegungen des Kopfes.

Man kann sich leicht vorstellen, daß ein so aufgebauter Drucker eine Menge Eigenintelligenz besitzen muß, um die Heizelemente zum richtigen Zeitpunkt ein- und auszuschalten. Für einen Mikrocomputer ist das (mit dem richtigen Programm!) aber kein Problem.

Es gibt auch Thermo-Druckköpfe, die, horizontal in einer Reihe angeordnet, derart viele Heizelemente besitzen, daß die Hin- und Herbewegung entfallen kann: Für jedes Zeichen (5×7-Matrix) einer Zeile müssen dann fünf Heizelemente vorhanden sein, in einer 40-Zeichen-Zeile also schon 200. Um die Ansteuerung zu vereinfachen, sind sie z.B. als 16×14-Matrix verschaltet, so daß „nur" noch 30 Anschlüsse nötig sind. Mit einer solchen Anordnung läßt sich eine recht hohe Druckgeschwindigkeit erzielen (etwa zehn Zeilen pro Sekunde).

Eine weitere Möglichkeit ist, nur sieben übereinander angeordnete Heizelemente zu verwenden. Der Kopf wird hier über die gesamte Zeilenbreite bewegt und erst dann das Papier weitertransportiert. Diese Anordnung ist z. B. bei schreibmaschinen-ähnlichen Terminals günstig, weil man hier ein eingetipptes Zeichen auch sofort drucken kann (bei den oben beschriebenen Verfahren kann ja erst dann gedruckt werden, wenn die Zeile vollständig eingegeben ist). Da der Kopf nur sieben Heizelemente hat, ist er recht preiswert herstellbar. Zudem ist die maximale Papierbreite nicht durch die Kopfkonstruktion begrenzt. Der Nachteil ist natürlich die geringe erreichbare Druckgeschwindigkeit von nur etwa 10...30 Zeichen pro Sekunde.

Thermo-Transfer-Drucker

Den Nachteil des Thermodruckers, teures Spezialpapier zu benötigen, zu vermeiden und gleichzeitig seine Vorteile wie Geräuscharmut und geringe Leistungsaufnahme beizubehalten, versucht man beim Thermotransfer-Druck. Dieses Verfahren kann man sich so vorstellen, daß die hitzeempfindliche Schicht hier nicht auf das Papier, sondern auf ein Farbband aufgebracht ist, von dem sie sich unter Hitzeeinwirkung löst und dann am darunterliegenden Papier haftet *(Abb. 2.1.1.2)*. Typisches Anwendungsbeispiel ist eine elektronische Reiseschreibmaschine *(Abb. 2.1.1.3)*: Wegen des geringen Stromverbrauchs ist Batteriebetrieb möglich, und Mitreisende werden wegen der Geräuscharmut nicht gestört.

Selbstverständlich sind auch hier abweichende Kopf-Konstruktionen denkbar, z. B. über die gesamte Zeilenbreite. Zwei Probleme seien nicht verschwiegen: Zum einen eignet sich das Verfahren nicht für jede Papiersorte, da bei zu rauher Oberfläche die vom Farbband abgelösten Zeichen nicht gut genug auf dem Papier haften. Zum anderen tritt das gleiche Datenschutz-Problem wie beim Karbon-Farbband elektrischer Schreibmaschinen (oder Typenraddrucker, siehe unten) auf: Auf dem „gebrauchten" Teil des Farbbands läßt sich alles nachlesen, was je damit geschrieben wurde ...

Nadeldrucker

Die heute meistverwendeten Drucker sind Nadeldrucker. Sie heißen so, weil sie mit Stahlnadeln arbeiten, die von Elektromagneten auf das Farbband gedrückt werden und so je einen Punkt auf dem Papier erzeugen *(Abb. 2.1.1.4)*.

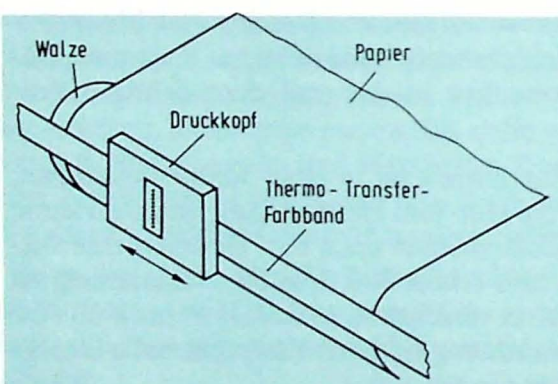


Abb. 2.1.1.2
Prinzip eines Thermo-Transfer-Druckers

Abb. 2.1.1.3 Elektronische Reiseschreibmaschine mit Thermo-Transfer-Kopf (EP-20 von Brother)

Üblicherweise ordnet man (ähnlich wie bei dem oben erwähnten Thermo-Transfer-Drucker) sieben oder mehr Nadeln übereinander an, so daß der Kopf jeweils eine Zeile druckt, wenn er horizontal über das Papier bewegt wird. Teilweise werden sogar bis zu 24 Nadeln verwendet, was ein Schriftbild ergibt, das kaum noch von einer normalen Schreibmaschine zu unterscheiden ist.

Prinzipiell ist damit (wie auch mit anderen Matrix-Druckverfahren) auch die Ausgabe von Grafik möglich. Dazu muß nur dafür gesorgt werden, daß über die Schnittstelle jede Nadel einzeln ansteuerbar ist. Allerdings läßt sich wegen des endlichen Nadel-Durchmessers nicht vermeiden, daß ein so gedruckter Kreis wie ein unregelmäßiges Vieleck aussieht — ein Problem, das man nur mit einem Plotter halbwegs beheben kann (s. u.).

Die meisten Nadeldrucker verfügen heute über mehrere, per Steuerbefehl umschaltbare Zeichensätze (amerikanisch, deutsch, skandinavisch usw.). Ferner ist es möglich, eines aus mehreren Druckverfahren auszuwählen, zum Beispiel:

Bidirektionaler Druck	Gedruckt wird in beiden Bewegungsrichtungen des Kopfes (Normalschrift)
Unidirektional-Druck	Nur bei Rechtsbewegung des Kopfes wird gedruckt. Besonders bei Grafik empfehlenswert, um Positionierungs-Ungenauigkeiten zu vermeiden
Fettdruck	Jedes Zeichen wird, um einen halben Punkt horizontal versetzt, zweimal gedruckt

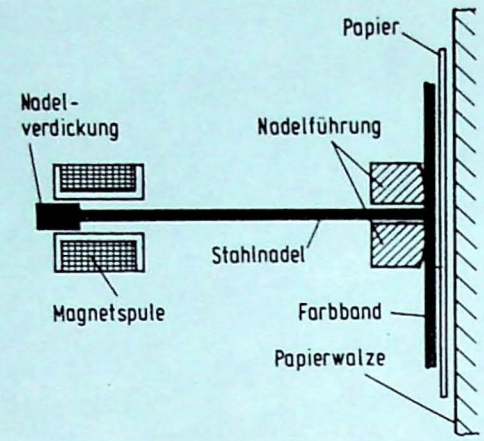


Abb. 2.1.1.4
Funktionsweise von
Nadeldruckern

Schönschrift	Jede Zeile wird, je um einen halben Punkt horizontal und vertikal versetzt, zweimal gedruckt. Aus einer 8×9-Matrix wird so eine 16×18-Matrix
Breitschrift	Jedes Zeichen wird doppelt breit geschrieben, indem jeder Matrixpunkt zweimal gedruckt wird
Kursivschrift	Die Zeichen werden leicht nach rechts geneigt ausgegeben

Selbstverständlich gibt es von all diesen Druckarten zahlreiche Kombinations-Möglichkeiten, wie etwa kursive und breite Schönschrift.

Leider sind die erforderlichen Steuerzeichen zum Umschalten von Zeichensätzen und Druckmodi von Hersteller zu Hersteller unterschiedlich. Dasselbe gilt für die Ansteuerung des Druckers zur Ausgabe von Grafiken. Es ist daher ratsam, hierfür das Handbuch genau zu studieren. Lediglich als Beispiel seien hier die Steuerzeichen des Druckers DX-2100 von Fujitsu hexadezimal aufgeführt, der mit dem überwiegend verwendeten Epson-Standard kompatibel ist.

```
0E                  Druck in doppelter Breite ein
1B 0E               ebenso
1B 57 01            ebenso
1B 57 00            Druck in doppelter Breite aus
14                  ebenso
0F                  komprimierte Schrift ein
1B 0F               ebenso
12                  komprimierte Schrift aus
1B 47               Doppelanschlag ein
1B 48               Doppelanschlag aus
1B 45               Fettschrift ein
1B 46               Fettschrift aus
1B 2D 01            Unterstreichen ein
1B 2D 00            Unterstreichen aus
1B 53 00            Hochgestellte Schrift
1B 53 01            Tiefgestellte Schrift
1B 54               Hoch- bzw. Tiefstellung aus
1B 34               Kursive Schrift ein
1B 35               Kursive Schrift aus
1B 70 01            Proportionalschrift ein
1B 70 00            Proportionalschrift aus
1B 21 n             Schriftarten setzen; Bits in n:
                    0=Elite, 1=Proportional, 2=komprimiert, 3=fett, 4=Doppel-
                    anschlag, 5=dopp. Breite, 6=kursiv, 7=unterstrichen
```

```
08                              1 Zeichen rückwärts
0D                              Wagenrücklauf (Carriage Return)
1B 50                           Schriftart Pica, 10 Zeichen pro Zoll (cpi)
1B 4D                           Schriftart Elite, 12 cpi
1B 6D                           Engschrift mit 15 cpi
1B 24 n1 n2                     Kopfposition=(256*n2+n1)/60 Zoll v.li.Rand
1B 5C n1 n2                     Kopfpos.=(256*n2+n1)/60 von der letzten Pos.
1B 20 n                         Zwischenraum zw. Zeichen = 0...63 (normal=0)
09                              Tabulator ausführen
1B 44 n1..nk 00                 Tabulatoren n1 bis nk setzen
0C                              Papiervorschub bis Blattende (Form Feed)
0A                              Papiervorschub um eine Zeile (Line Feed)
1B 4A n                         Papiervorschub um n/216 Zoll
1B 6A n                         Negativer Papiervorschub um n/216 Zoll
1B 30                           Zeilenvorschub auf 1/8 Zoll setzen
1B 31                           Zeilenvorschub auf 7/72 Zoll setzen
1B 32                           Zeilenvorschub auf 1/6 Zoll setzen
1B 33 n                         Zeilenvorschub auf n/216 Zoll setzen
1B 41 n                         Zeilenvorschub auf n/72 Zoll setzen
0B                              Vertikalen Tabulator ausführen
1B 42 n1..nk 00                 Vertikale Tabulatoren n1 bis nk setzen
1B 62 n n1..nk 00               Tabs in Vertikal-Format-Kanal n setzen
1B 2F n                         Vertikalen Format-Kanal n (0...7) anwählen
1B 51 n                         Linken Rand = n setzen
1B 6C n                         Rechten Rand = n setzen
1B 4E n                         Perforationen um n Zeilen überspringen
1B 4F                           Perforationen nicht mehr überspringen
1B 43 n                         Seitenlänge = n Zeilen setzen
1B 43 00 n                      Seitenlänge = n Zoll setzen
1B 52 n                         Nationalen Zeichensatz auswählen; n:
                                0=USA, 1=Frankreich, 2=Deutschland, 3=
                                England, 4=Dänemark, 5=Schweden, 6=Italien,
                                7=Spanien, 8=Japan, 9=Norwegen, 10=Dänem.II
1B 2A m n1 n2 p1..pk            8-Punkte-Grafikmodus m setzen (0...8),
                                Druck ab Pos. 0 bis 256*n1+n2, p=Datenbytes
1B 5E m n1 n2 p1..pk            9-Punkte-Grafikmodus m setzen (doppel-
                                te Zahl von Datenbytes p gegenüber 8 Punkten)
1B 4B n1 n2 p1..pk              Grafikmodus "einfache Dichte" setzen
1B 4C n1 n2 p1..pk              Grafikmodus "doppelte Dichte" setzen
1B 59 n1 n2 p1..pk              Grafikmodus mit doppelter Dichte und
                                doppelter Geschwindigkeit setzen
1B 5A n1 n2 p1..pk              Grafikmodus "vierfache Dichte" setzen
1B 3F n m                       Grafikmodus n zu Modus m konvertieren
1B 25 00 00                     ROM-Zeichensatz wählen
1B 25 01 00                     RAM-Zeichensatz wählen (Download char. set)
1B 25 02 00                     Near-Letter-Zeichensatz (NLQ, Schönschrift)
1B 78 01                        ebenso
1B 78 00                        Draft-Zeichensatz wählen (Schnellschrift)
1B 3A 00 00 00                  ROM-Zeichensatz ins RAM kopieren
1B 26 00 n m a p1..p10 00       Draft-Download-Zeichen definieren
1B 26 01 n m a1 a2 p1..p47 00   NLQ-Download-Zeichen definieren
11                              Drucker selektieren
13                              Drucker deselektieren
07                              Piepton erzeugen
18                              Vorherige Daten löschen
7F                              Pufferspeicher löschen
1B 3C                           Druckkopf in Ruhestellung fahren
1B 3D                           Bit 7 (MSB) = 0 setzen
1B 3E                           Bit 7 (MSB) = 1 setzen
1B 23                           Bit 7 (MSB) wieder wie Eingangsdaten
1B 38                           Papierende-Detektor desaktivieren
1B 39                           Papierende-Detektor aktivieren
1B 69 01                        View-Mode einschalten (sofort drucken)
1B 69 00                        View-Mode ausschalten
1B 36                           Steuerzeichen-Druck (Codes 80...9F, MSB=1!)
1B 37                           Steuerzeichen-Druck ausschalten
1B 49 01                        Undefinierte Codes drucken
1B 49 00                        Undefinierte Codes nicht drucken
1B 73 01                        Mit halber Geschwindigkeit drucken
1B 73 00                        Mit normaler Geschwindigkeit drucken
```

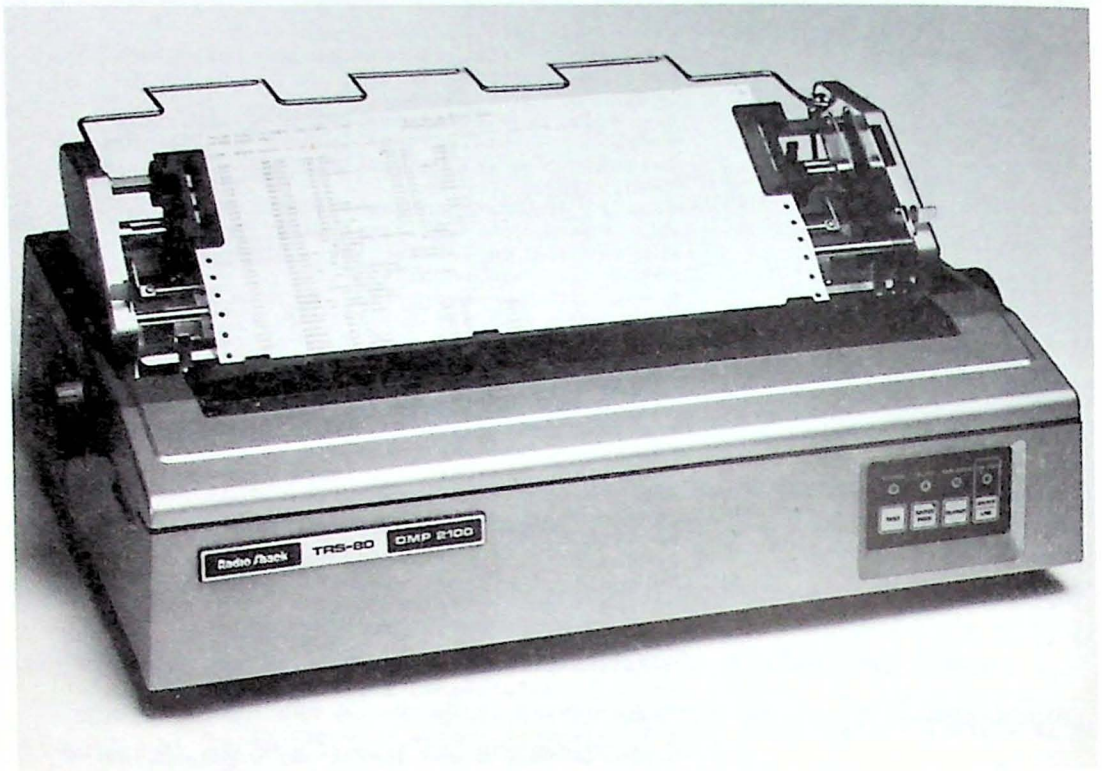


Abb. 2.1.1.5 Nadeldrucker mit Traktor-Führung für seitlich gelochtes Papier (Tandy)

```
1B 40              Drucker rücksetzen (Puffer nicht gelöscht)
1B 55 01           Unidirektionaler Druck
1B 55 00           Bidirektionaler Druck
1B 56 n d1..dm     Daten d1...dm n-mal drucken
1B 72 n            Farbe n setzen (nur bei Farbdruckern); n:
                   0=schwarz, 1=magenta, 2=cyan, 3=violett,
                   4=gelb, 5=orange, 6=grün
```

Normalerweise wird eine Zeile erst dann ausgedruckt, wenn sie mit einem CR- oder LF-Zeichen abgeschlossen ist. Da manche Computer am Zeilenende kein LF-Zeichen generieren, lassen sich die meisten Drucker z.B. mit einem Schalter oder einer Drahtbrücke namens „Auto LF" so einstellen, daß ein Zeilenvorschub auch ohne LF automatisch bei einem CR-Zeichen stattfindet.

Für den Transport und die Zuführung des Papiers gibt es unterschiedliche Möglichkeiten. Professionelle Nadeldrucker *(Abb. 2.1.1.5)* lassen sich oft für die eine oder andere aus- und nachrüsten:

Normalerweise wird das Papier wie bei einer Schreibmaschine durch eine Walze transportiert, auf die es von Andruckrollen gepreßt wird, um ein unabsichtliches Verrutschen zu vermeiden. Für einzelne Blätter ist dies durchaus ausreichend.

Will man aber z.B. Endlos-Formulare (aneinandergereihte Blätter, die durch eine Perforation trennbar sind) oder Endlos-Etikettenträger bedrucken, so müssen

auch nach vielen Zeilen die vorgesehenen Schriftfelder noch exakt getroffen werden. Dazu wäre der unvermeidliche Schlupf einer reinen Walzenführung zu groß.

In diesen Fällen, aber auch bei dem typischen Endlos-Computerpapier, verwendet man einen „Traktor" — hier nicht etwa ein landwirtschaftliches, sondern ein Papier-Transportmittel, auch Stachelwalze genannt. Die Walze ist hier über Riemen, Zahnräder oder Ketten mit zwei Stachelrädern verbunden, die in den gelochten Rand des hierfür vorgesehenen Papiers greifen und sozusagen gewaltsam für einen präzisen Transport sorgen. Die rechte Stachelwalze ist manchmal verschiebbar, damit Papier unterschiedlicher Breite eingespannt werden kann (üblich: 254 mm = 10 Zoll; 432 mm = 17 Zoll; 210 mm = A4; 297 mm = A3).

Eine weitere Drucker-Option ist die „automatische Einzelblatt-Zuführung". Sie ermöglicht (ähnlich wie bei einem Fotokopierer), serienmäßig viele Blätter ohne Lochrand zu bedrucken, ohne jedes einzeln einlegen zu müssen. Das ist z.B. beim Schreiben von Serienbriefen nützlich. Selbstverständlich gibt es solche Zusatzeinrichtungen nicht nur für Nadel-, sondern auch für Typenrad- und Typenkorb-Drucker, die später besprochen werden.

Weitere Matrix-Druckverfahren

Beim sogenannten Tintenstrahl-Drucker wird eine spezielle, schnell trocknende Tinte von einem Ultraschall-Zerstäuber punktförmig auf das Papier gesprüht. Der dabei erzielbare Kontrast ist allerdings von der Papiersorte abhängig; je mehr Tinte das Papier aufsaugt, desto blasser wird der Ausdruck. Eine glatte Papieroberfläche führt zum besten Ergebnis. Der wesentlichste Vorteil dieses Prinzips ist, daß der Druckvorgang nahezu lautlos ist. Die erzielbare Geschwindigkeit ist mit der eines Nadeldruckers vergleichbar (etwa 200 Zeichen pro Sekunde).

Wem es auf sehr viel höhere Druckgeschwindigkeiten ankommt, kann einen Laserdrucker verwenden. Sein Prinzip ähnelt ein wenig dem eines Fotokopiergeräts: In seinem Inneren sitzt eine rotierende Trommel, die mit einem fotoempfindlichen, halbleitenden Material (Selen oder Silizium) beschichtet ist. Bei Belichtung (im Fotokopierer durch das Original-Schriftstück, im Laser-Drucker zeilenweise durch einen Laserstrahl) kann sich punktförmig eine hohe Spannung auf der Trommel aufbauen, so daß vorbeibewegte Farbstoff-Teilchen an ihr haften bleiben. Wie beim Offsetdruck kann nun die Trommel auf einen Papierbogen abgewälzt werden, so daß der Farbstoff auf das Papier übertragen wird. Sowohl die Belichtung durch den Laser als auch das Abrollen des Farbstoffs können sehr rasch erfolgen, und ein ganzer Papierbogen kann so innerhalb von fünf Sekunden bedruckt werden.

2.1.2 Typenrad- und Typenkorb-Drucker

Die ersten Schreibmaschinen und elektromechanischen Fernschreibgeräte arbeiteten mit Typenhebeln. Dabei sind auf einem Hebel jeweils zwei Zeichen spiegelverkehrt untergebracht, z.B. a und A (Zeichen ohne und mit Shift-Taste). Leider ist dieses Prinzip für Drucker allerdings wenig geeignet: Wegen der recht großen trä-

gen Masse eines solchen Hebels ist zu seinem Anschlagen eine hohe Energie nötig, die erreichbare Druckgeschwindigkeit ist gering und eine hohe Lärmentwicklung unvermeidbar. Typenhebel-Drucker spielen heute deshalb als Computer-Ausgabegeräte so gut wie keine Rolle.

Verkleinert man jedoch die Typenhebel und ordnet sie sternförmig um eine Achse an, so gelangt man zum Typenrad. Im Gegensatz zu Schreibmaschinen und Fernschreibern ist auf einer Type allerdings nur ein einziges Zeichen untergebracht, für „a" und „A" sind also getrennte Typen vorhanden. Dies macht einen Höhenversatz von Walze oder Anschlag-Mechanismus überflüssig *(Abb. 2.1.2.1)*. Eine Sonderform, der Typenkorb, entsteht, indem man die federnden Typen zu einer korbähnlichen Form aufbiegt; der Vorteil ist die geringere nötige Beschleunigung zum Drehen, der Nachteil, daß sich die Typen bei hoher Drehgeschwindigkeit unter dem Einfluß der Fliehkraft ein wenig von selbst nach außen biegen. Überwiegend werden heute, auch wegen der einfacheren Fertigung, Typenräder verwendet.

Die Druckerlogik muß nun zum Druck eines bestimmten Zeichens dafür sorgen, daß sich die zugehörige Type möglichst schnell an den Anschlagpunkt bewegt. So-

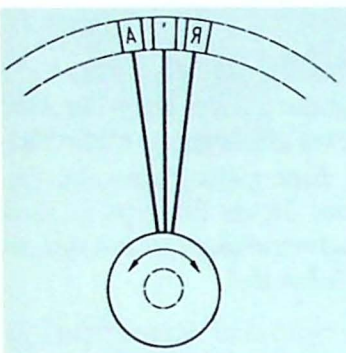


Abb. 2.1.2.1
Bei Typenraddruckern sind die einzelnen Zeichen auf einem schnell drehbaren Rad untergebracht

Abb. 2.1.2.2 Typenrad- oder Daisy-Wheel-Drucker (DW-410 von Tandy)

bald das geschehen ist, erfolgt der Anschlag durch einen kleinen Elektromagnet. Da in normalem Text manche Buchstaben wie e, t und s wesentlich häufiger vorkommen als andere, sind die Schriftzeichen nicht einfach z.B. in ASCII-Reihenfolge (a, b, c, d usw.) auf dem Typenrad untergebracht, sondern so, daß bei statistischem Text der durchschnittliche nötige Drehwinkel des Rades zum Anschlag des nächsten Zeichens möglichst gering ist. Wenn man so auf das Typenrad sieht, daß die Zeichen spiegelverkehrt erscheinen, lautet eine typische Reihenfolge im Uhrzeigersinn (Dyneer DW-16):

05678<ä$?"%ö+Äç'&;_> =*()ß§¬Öü/#Ü!ml9gvb-ucnroeaithsfpkywj
x,W.MZBFCA`RSETHONILDUGYPQKJV:X1234

Das erste Zeichen, die Null, steht hierbei über dem Markierungsloch, das nahe der Drehachse für eine eindeutige Einsetzrichtung sorgt. Die Zeichen-Reihenfolge entstand übrigens nicht nach deutschem, sondern amerikanischem Standard-Text (sog. Shannon-Text; die Häufigkeits-Verteilung ist natürlich deutlich von der Sprache abhängig).

Im Gegensatz zu Matrixdruckern kann man bei Typenrad-Druckern keine feste Geschwindigkeit angeben. Schafft ein Drucker z.B. pro Sekunde 40 Zeichen, wenn er immer nur den gleichen Buchstaben anschlägt, so kommt er bei Shannon-Text vermutlich nur noch auf etwa 30 und bei deutschem Text vermutlich auf noch weniger.

Elektrische Typenrad-Schreibmaschinen und preiswerte Typenrad-Drucker können bei durchschnittlichem Text etwa 14 bis 20 Zeichen pro Sekunde ausgeben, typische Profi-Ausführungen etwa 30...40 *(Abb. 2.1.2.2)* und die allerschnellsten Geräte bis zu 80, also nur etwa ein Viertel von Nadeldruckern. Außerdem ist ein Wechsel der Schriftart nicht per Steuerbefehl, sondern nur durch Austausch des Typenrades machbar. Lediglich der Zeilen- und Zeichenabstand ist hier steuerbar.

Dem steht als Hauptvorteil die sehr gute Schriftqualität gegenüber, die ein Nadeldrucker auch im Schönschrift-Modus kaum erreichen kann. Ferner ist die Verwendung sehr kontrastreicher Karbon-Einweg-Farbbänder möglich.

Wie bei Nadeldruckern läßt sich eine Reihe von Funktionen vom Computer aus fernsteuern. Die folgende Tabelle nennt beispielhaft die Steuerzeichen-Sequenzen des Druckers DW-16 von Dyneer; sie entsprechen dem Diablo-Standard:

Zeichenfolge	Hex	Funktion
BEL	07	Piepton
BS	08	1 Zeichen zurück
CR	0D	Wagenrücklauf
ESC 0	1B 30	Rechter Rand = Kopfposition
ESC 1	1B 31	Hor.Tab. = Kopfposition
ESC 2	1B 32	Alle Tabulatoren löschen
ESC 3	1B 33	Grafik-Modus ein
ESC 4	1B 34	Grafik-Modus aus
ESC 5	1B 35	Vorwärts drucken
ESC 6	1B 36	Rückwärts drucken (CR = Beenden)
ESC 8	1B 38	Tab. an Kopfpos. löschen

ESC 9	1B 39	Linker Rand = Kopfposition
ESC D	1B 44	Halbe Zeile nach unten
ESC U	1B 55	Halbe Zeile nach oben
ESC Y	1B 59	Sonderzeichen: c
ESC Z	1B 5A	Sonderzeichen: ¬
ESC LF	1B 0A	Eine Zeile nach unten
ESC HT n	1B 09 n	Kopf an n-te Position der Zeile bringen
ESC VT n	1B 0B n	Kopf n Zeilen vom Seitenanfang nach unten
ESC RS n	1B 1E n	Zeilenabstand in 48stel Zoll setzen
ESC US n	1B 1F n	Zeichenabstand in 120stel Zoll setzen
ESC CR P	1B 0D 50	Alle Parameter auf Normalwerte setzen
FF	0C	Vorschub zum Beginn der nächsten Seite
HT	09	Zum nächsten Tabulator gehen
LF	0A	Zur nächsten Zeile gehen
Space	20	Ein Zeichen vorwärts (Leerraum)

Um den an sich eher langsamen Typenrad-Drucker etwas schneller zu machen, gibt es mehrere Möglichkeiten. Zunächst einmal kann man wie bei Nadeldruckern aufeinanderfolgende Zeilen abwechselnd von links und von rechts drucken (Voraussetzung ist natürlich, daß der Drucker jeweils eine komplette Zeile speichern kann). Dann ist es möglich, aufeinanderfolgende Leerräume mit einer höheren Geschwindigkeit als beim normalen Druck-Vorschub zu überspringen, ebenso Leerzeilen oder Zeilen, die ohnehin nur Leerräume enthalten. All diese Möglichkeiten gibt es inzwischen zwar auch bei Nadeldruckern; bei Typenrad-Druckern sind sie aber von besonderer Bedeutung.

Die Ausgabe von Grafiken ist mit Typenrad-Druckern natürlich nur sehr begrenzt möglich, beispielsweise durch schrittweises Positionieren des Kopfes und Druck des Punkt-Zeichens.

2.1.3 Plotter

Speziell für die Ausgabe von Zeichnungen ist der Plotter gedacht. Er setzt Zeichen und Grafiken nicht aus einzelnen Punkten, sondern aus Linien zusammen, deren Endpunkte ihm vom Computer übergeben werden. *Abb. 2.1.3.1* zeigt deutlich, daß eine so entstandene Zeichnung nichts zu wünschen übrig läßt.

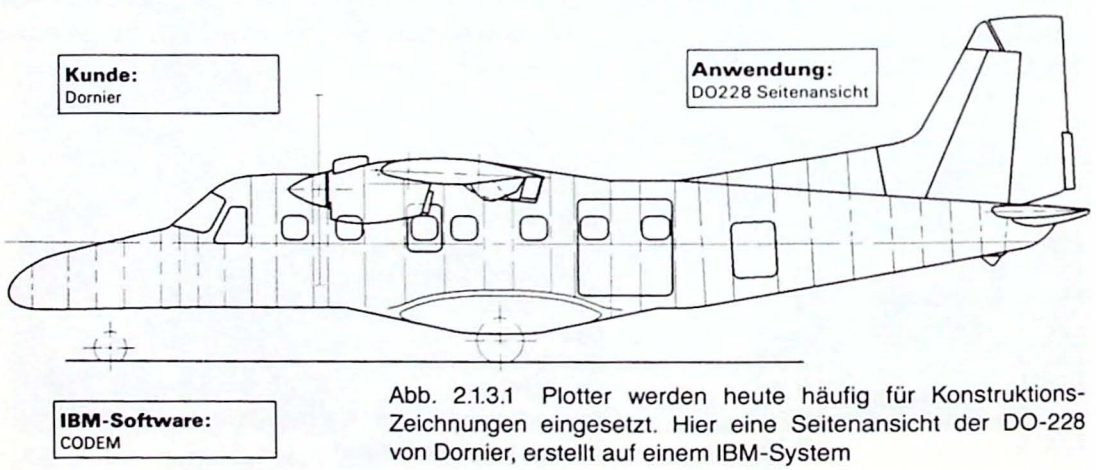


Abb. 2.1.3.1 Plotter werden heute häufig für Konstruktions-Zeichnungen eingesetzt. Hier eine Seitenansicht der DO-228 von Dornier, erstellt auf einem IBM-System

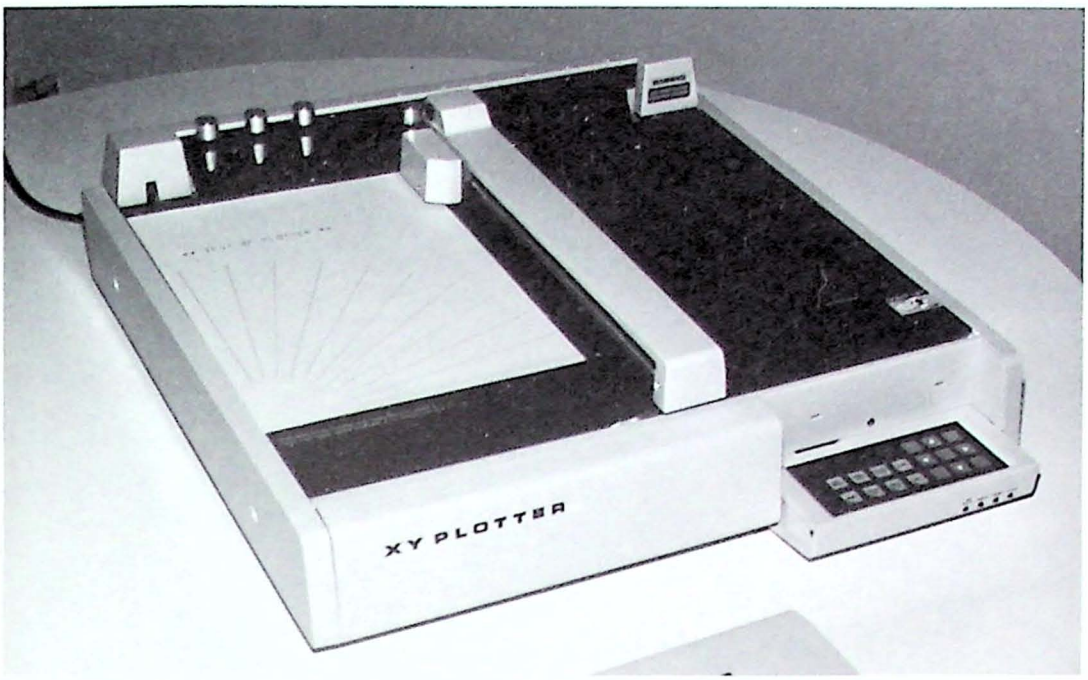


Abb. 2.1.3.2 Vierfarben-Plotter mit herausklappbarem Bedienfeld (Graphtec). Das Papier wird elektrostatisch festgehalten

Zum Schreiben dient meist ein gewöhnlicher Filzstift, der in X- und Y-Richtung über das flach liegende Papier geführt wird. Damit das Papier nicht verrutschen kann, wird es entweder am Rand festgeklemmt oder elektrostatisch festgehalten, indem die Unterlage auf eine im Gerät erzeugte hohe Spannung aufgeladen wird (die allerdings wegen der geringen Strom-Belastbarkeit völlig ungefährlich ist). In X-Richtung (also von links nach rechts) wird durch einen Schrittmotor ein Schlitten gefahren, auf dem sich in Y-Richtung der Schreibstift hin- und herbewegen läßt *(Abb. 2.1.3.2)*. Ein eigener Elektromagnet senkt den Stift erst dann auf die Papieroberfläche, wenn gezeichnet werden soll. Eine trickreiche Mechanik erlaubt sogar das Anfertigen mehrfarbiger Grafiken, indem sich der Schlitten aus einer Ablage (im Bild links hinten) verschiedenfarbige Stifte holen kann.

Eine andere Methode zeigt *Abb. 2.1.3.3:* Hier wird das Papier selbst in Y-Richtung und der Schreibstift in X-Richtung bewegt. Das kostet weniger Energie, weil sich ein Blatt Papier viel leichter bewegen läßt als der schwere Schlitten. Allerdings muß eine gute Papierführung dafür sorgen, daß auch nach einigen Hin- und Herbewegungen kein Schlupf auftritt. Hierzu sind links und rechts geriffelte Andruckrollen vorhanden, die so auf das Papier drücken, daß es selbst eine feine Riffelung erhält und nicht verrutschen kann.

Eine Abart davon ist der Trommelplotter, wie er für kleine, tragbare Computer gern verwendet wird *(Abb. 2.1.3.4)*. Hier sitzen z. B. vier verschiedenfarbige Kugelschreiber-Minen in einer kleinen Trommel, die von einem kleinen Motor gedreht werden kann, so daß nur der Stift mit der gewünschten Farbe über dem Papier

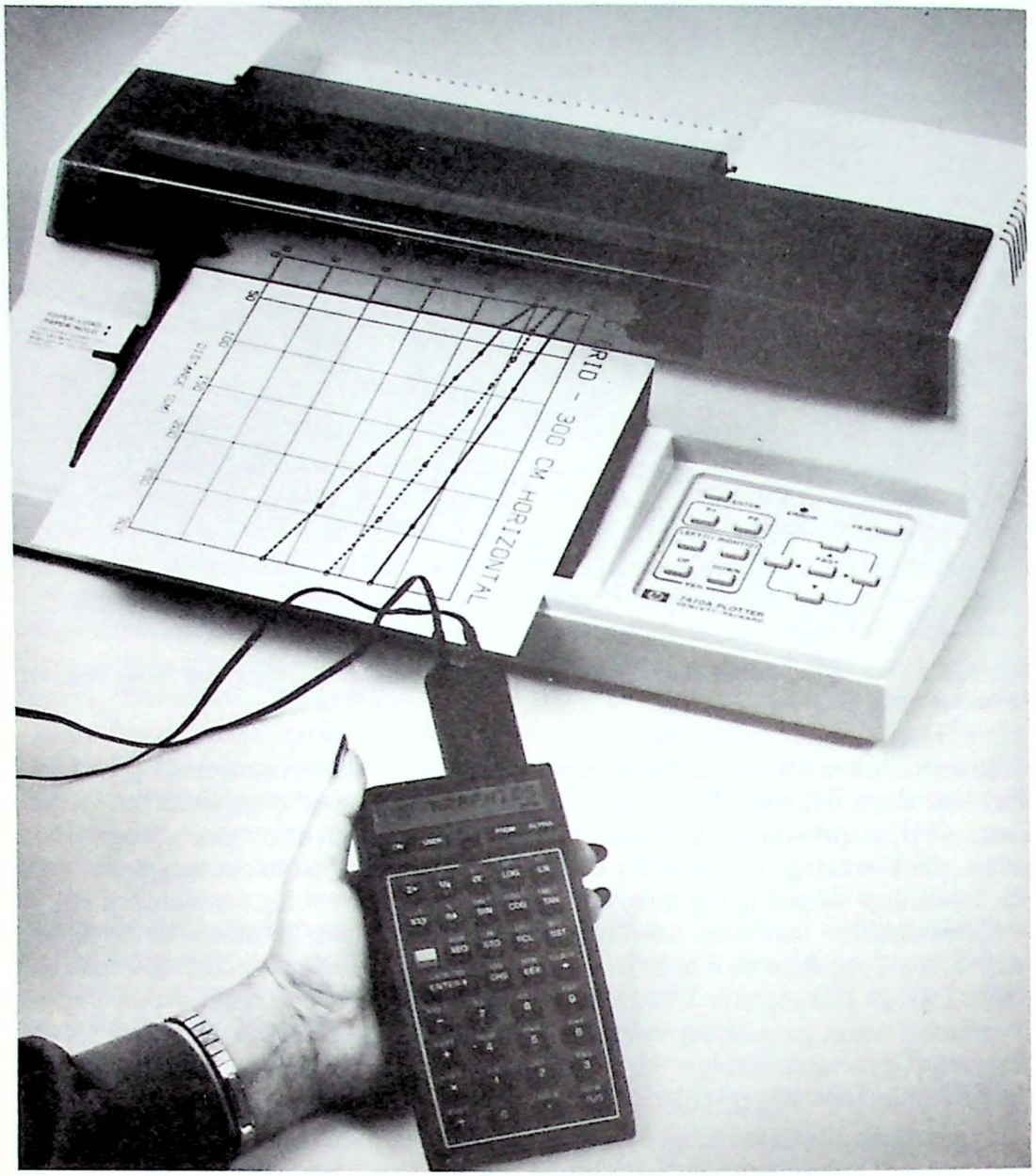


Abb. 2.1.3.3 Bei diesem Plotter wird das Papier hin- und herbewegt. Der Schreibstift muß daher nur in einer Richtung, nämlich quer dazu, beweglich sein (Hewlett-Packard)

steht. Ein zusätzlicher Elektromagnet übernimmt das Absenken des Stiftes, sobald eine Linie gezeichnet werden soll. Die Minentrommel bewegt sich zum Plotten in X-Richtung auf das Papier in Y-Richtung; allerdings ist hier eine Rückwärts-Bewegung nur ein paar Zentimeter erlaubt, da sich das Papier sonst im Rollenfach verheddern kann. Ferner ist die Wiederkehrgenauigkeit eines solchen Plotters ohnehin nicht allzu hoch.

Abb. 2.1.3.4 Links neben Olivettis tragbarem Computer M-10 ist hier ein kleiner Plotter zu sehen, der mit einer Kugelschreiber-Trommel arbeitet

Unabhängig vom mechanischen Prinzip eignen sich Plotter auch zur Ausgabe von Schriftzeichen, die wie Grafiken nicht aus Punkten, sondern aus Linien zusammengesetzt werden. Die so erreichbare Schreibgeschwindigkeit ist (je nach Schriftgröße) mit etwa drei bis zehn Zeichen pro Sekunde recht gering. Die Fähigkeit, auch Schrift auszugeben, macht einen Plotter daher nicht zum Drucker-Ersatz, sondern dient nur der Beschriftung von Zeichnungen. Vorteilhaft ist dabei, daß die Schrift in unterschiedlichen Größen und Schreibrichtungen geplottet werden kann, also z. B. auch in Richtung der Y-Achse.

Die Kommando-Syntax, die der Computer dem Plotter über die Schnittstelle zu übergeben hat, ist sehr herstellerspezifisch. Das gilt auch für die Eigenintelligenz des Plotters. Während einige Typen wirklich nur eine Linie von einem Koordinaten-Punkt zum anderen zeichnen können, ist mit manchen z. B. das Zeichnen von Kreisen möglich, indem man nur die Mittelpunkt-Koordination und den Radius übergibt. Besonders intelligente können sogar selbständig Flächen schraffieren; da sie sich hierzu die Eckpunkte „merken" müssen, ist dies allerdings nur bis zu einer bestimmten Zahl von Eckpunkten möglich.

Plotter, die über eine bidirektionale V.24-Schnittstelle oder einen IEC-Bus-Anschluß verfügen, erlauben es manchmal auch, daß der Computer die aktuellen Koordinaten des Schreibstiftes abfragt. Bei einer Centronics-Schnittstelle ist dieser Komfort leider nicht realisierbar, weil der Datentransfer nur vom Computer zum Plotter stattfinden kann.

Sowohl Filzstifte als auch Kugelschreiber-Minen sollte man aus Plottern herausnehmen und ihre Spitzen luftdicht verschließen, wenn man das Gerät längere Zeit nicht verwendet. Andernfalls sind sie nach kurzer Zeit nicht mehr zu gebrauchen, weil die Schreibspitzen eintrocknen.

2.1.4 Sprachausgabe

Ein noch selten verwendetes, nichtsdestotrotz sehr interessantes Ausgabe-Verfahren bei Mikrocomputern ist die Sprachausgabe. Dafür gibt es unterschiedliche Möglichkeiten.

Am naheliegendsten ist es natürlich, die einmal vor einem Mikrofon gesprochenen Worte einem Analog/Digital-Wandler zuzuleiten, dessen digitales Ausgangssignal im Mikrocomputer zu speichern und später bei Bedarf über einen D/A-Wandler im gleichen Takt wieder auszugeben.

Bedauerlicherweise sagt die Theorie von Shannon, daß die nötige Abtastfrequenz dabei mindestens doppelt so hoch wie die höchste Signalfrequenz sein muß. Mit anderen Worten: Für eine bei Sprache sinnvolle Maximal-Signalfrequenz von 3 kHz ist die Speicherung von mindestens 6000 Abtastwerten pro Sekunde nötig. Bei einem 8-Bit-A/D-Wandler sind das pro Sekunde 6000 × 8 Bit, also knapp 6 KByte (1 KByte = 1024 Byte, 1 Byte = 8 Bit). Wo es auf hohe Qualität ankommt (PCM-Aufzeichnung, automatische Telefon-Auskunft usw.), wird dieses Prinzip tatsächlich eingesetzt.

Erstaunlicherweise läßt sich der A/D- bzw. D/A-Umsetzer einsparen, und trotzdem bleibt die Sprache noch einigermaßen verständlich *(Abb. 2.1.4.1)*. Nach dem Mikrofon-Vorverstärker sorgt ein Begrenzer für die Generierung reiner Digitalwerte, die dann bitseriell im Computer abgelegt werden. Da bei der Wiedergabe dieses amplituden-begrenzten Signals erhebliche Oberton-Anteile im Bereich über 3 kHz auftreten, ist anstelle eines D/A-Wandlers ein 3-kHz-Tiefpaß sinnvoll, der nur den Sprach-Frequenzbereich durchläßt. Erfahrungen haben gezeigt, daß mit diesem Prinzip bei 8 kHz Abtastfrequenz eine noch ausreichende Verständlichkeit erzielbar ist, was einem Speicherplatz-Bedarf von 8000 Bit/s bzw. knapp 1 KByte/s entspricht. Dieser Wert läßt sich noch reduzieren, wenn man Sprachpausen bzw. sich wiederholende Bytes codiert in nur ein oder zwei Bytes abspeichert. Dieses Prinzip ist z.B. zur automatischen Rufzeichen-Einblendung bei Funkverbindungen geeignet.

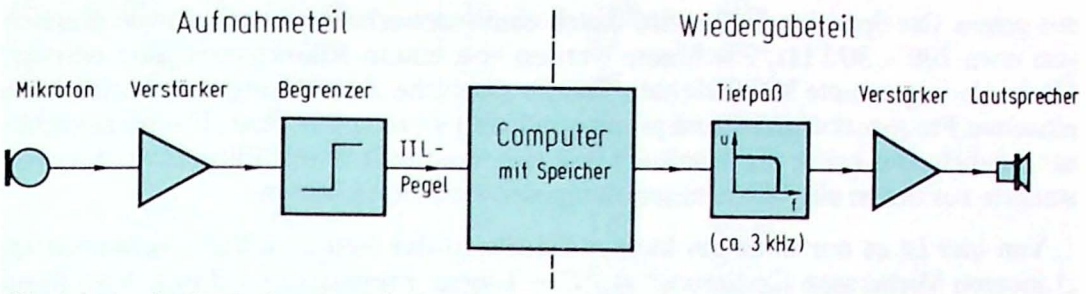


Abb. 2.1.4.1 Auch mit einer so einfachen Anordnung läßt sich eine ausreichende Verständlichkeit bei der Sprachwiedergabe erzielen

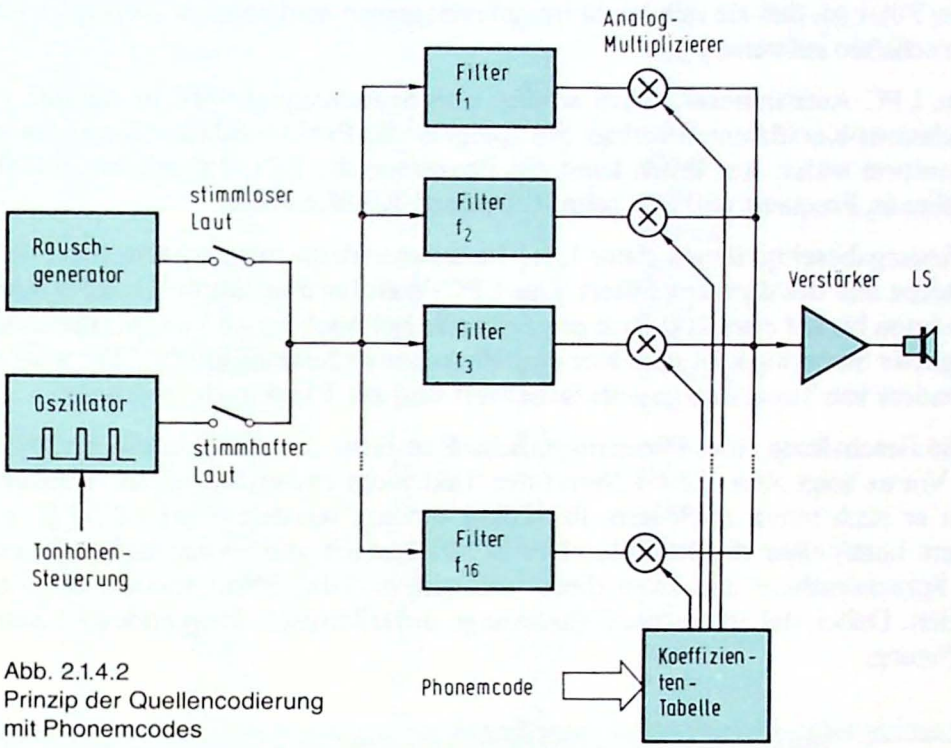


Abb. 2.1.4.2
Prinzip der Quellencodierung
mit Phonemcodes

Bei beiden bisher genannten Verfahren ist der Speicherplatz-Bedarf noch relativ hoch. Man hat sich daher Gedanken gemacht, wie durch elektronische Nachbildung des menschlichen Sprachtrakts eine Reduzierung der Informationsmenge erreicht werden kann.

Einen möglichen Ausweg zeigt *Abb. 2.1.4.2* mit dem Prinzip der Quellen- bzw. Wellenform-Codierung. Die Überlegung hierbei ist, daß sich zwar die Amplitude der Sprache rasch und häufig ändert, nicht aber ihre spektrale Verteilung, d.h. ihre Frequenz-Zusammensetzung. Zerlegt man das Sprachspektrum in möglichst viele einzelne Bereiche und überträgt bzw. speichert lediglich die Amplituden-Änderungen für jeden dieser „Kanäle", so erreicht man eine recht gute Qualität bei 1 KByte/s, ohne daß (wie im letzten Beispiel) die Amplituden-Änderungen verlo-

ren gehen. Die Sprachmelodie wird durch einen steuerbaren Oszillator im Bereich von etwa 200...300 Hz, Zischlaute werden von einem Rauschgenerator erzeugt. Über eine sogenannte Koeffizienten-Tabelle (logische Zuordnung der Amplituden einzelner Frequenzbereiche zum jeweiligen Laut) ist so sogar eine „Phonemsynthese" möglich. Das heißt, die Schaltung läßt sich unmittelbar mit Phonem-Codes ansteuern, aus denen alle Worte zusammengesetzt werden können.

Von hier ist es nur noch ein kleiner Schritt zu der heute vielfach verwendeten „Linearen Vorhersage-Codierung" (LPC = Linear Predictive Coding). Man kann sich dieses Prinzip so vorstellen, daß die analogen Filter in Form eines einzigen, komplexen digitalen Filters zusammengefaßt werden *(Abb. 2.1.4.3)*. Der Vorteil digitaler Filter ist, daß sie sich leicht integrieren lassen und sehr gut reproduzierbare Eigenschaften aufweisen.

Die LPC-Aufnahmeseite sieht so aus: Das Sprachsignal wird in digitale Autokorrelations-Koeffizienten zerlegt. Sie spiegeln die Periodendauer der analysierten Wellenform wider. Aus ihnen kann ein Prozessor die LPC-Parameter errechnen: Amplitude, Frequenz und z. B. zehn Reflexions-Koeffizienten.

Wiedergabeseitig dienen diese LPC-Parameter dann zur Ansteuerung der Oszillatoren und des digitalen Filters. Das LPC-Verfahren erlaubt eine Informations-Reduktion bis auf etwa 200 Byte pro Sekunde bei noch guter Verständlichkeit. Bei geeigneter Steuerlogik ist auch hier eine Phonemsynthese möglich. LPC wurde insbesondere von Texas Instruments favorisiert und zur Marktreife gebracht.

Die Beschaltung eines Phonemsynthese-Bausteins des amerikanischen Herstellers Votrax zeigt *Abb. 2.1.4.4.* Wenn der Takt nicht extern zugeführt werden soll, kann er auch intern mit einem RC-Glied erzeugt werden *(Abb. 2.1.4.5);* er bestimmt unmittelbar die Tonhöhe. Der SC-01 enthält alle wesentlichen Bausteine zur Sprachsynthese und kann direkt mit einem 5-Bit-Phonemcode angesteuert werden. Dabei stehen folgende (allerdings amerikanisch klingenden) Laute zur Verfügung:

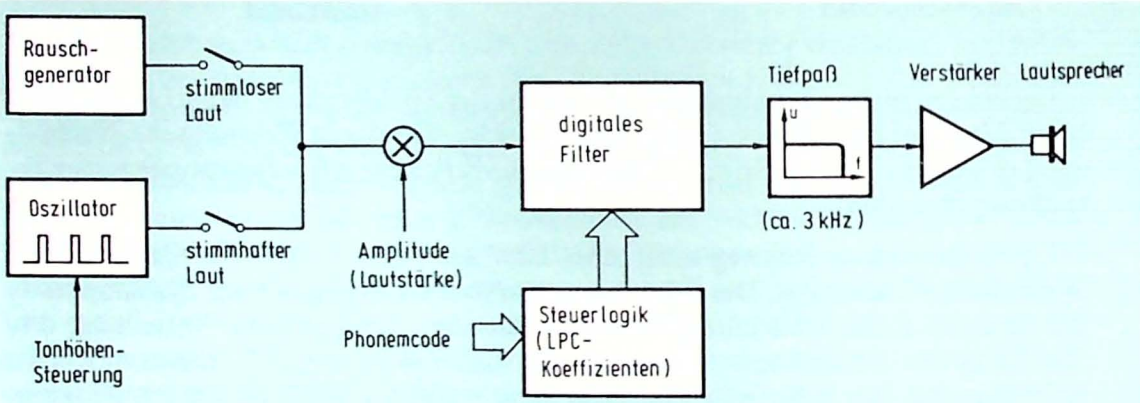


Abb. 2.1.4.3 Funktionsweise einer Sprachausgabe nach dem LPC-Verfahren (Linear Predective Coding)

Code	Länge	Beispiel	Code	Länge	Beispiel
00	59	jacke!t	20	185	da!y
01	71	e!nlist	21	65	da!y
02	121	hea!vy	22	80	y!ard
03	47	Pause	23	47	missio!n
04	47	butt!er	24	250	mo!p
05	71	ma!de	25	103	p!ast
06	103	ma!de	26	185	co!ld
07	90	az!ure	27	185	pi!n
08	71	ho!nest	28	185	mo!ve
09	55	inhibi!t	29	103	any!
0A	80	i!nhibit	2A	71	t!ap
0B	121	inhi!bit	2B	90	r!ed
0C	103	m!at	2C	185	mee!t
0D	80	sun!	2D	80	w!in
0E	71	b!ag	2E	185	da!d
0F	71	v!an	2F	103	a!fter
10	71	ch!ip	30	90	sa!lty
11	121	sh!op	31	71	a!bout
12	71	z!oo	32	103	u!ncle
13	146	la!wful	33	185	cu!p
14	121	thing!	34	80	fo!r
15	146	fa!ther	35	121	aboa!rd
16	103	loo!king	36	59	you!
17	185	boo!k	37	90	you!
18	103	l!and	38	80	th!e
19	80	trick!	39	71	th!in
1A	47	judg!e	3A	146	bi!rd
1B	71	h!ello	3B	185	ge!t
1C	71	g!et	3C	121	be!
1D	103	f!ast	3D	250	ca!ll
1E	55	paid!	3E	185	Pause
1F	90	pass!	3F	47	Pause

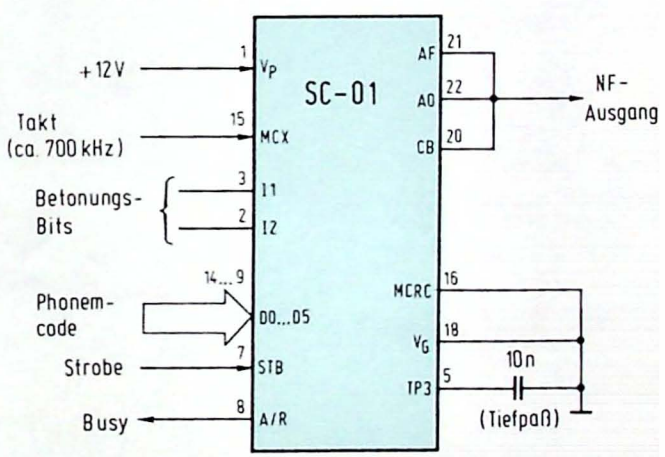


Abb. 2.1.4.4
Anschlußschema des integrierten Sprachsynthese-Bausteins SC-01 von Votrax

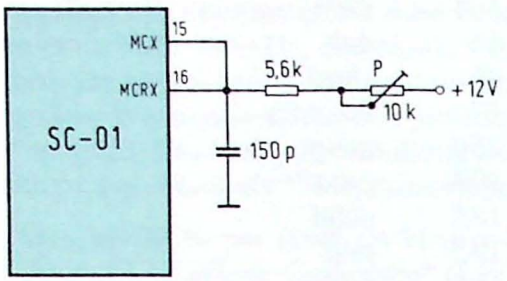


Abb. 2.1.4.5
Der Takt kann beim SC-01 mit Hilfe weniger Bauelemente auch vom Chip selbst erzeugt werden

Die Ausrufezeichen stehen hier jeweils hinter dem erzeugten Phonem. Es ist leicht zu erkennen, daß z.B. allein für das „e" eine ganze Reihe unterschiedlicher Phonem-Versionen nötig ist, um den Klang stets einigermaßen richtig zu treffen.

Außer den Phonemcode-Eingängen gibt es noch zwei Betonungs-Eingänge, mit denen die Sprachmelodie gesteuert werden kann. Inzwischen gibt es vom Hersteller Votrax übrigens auch eine Baustein-Version namens SC-02, die sich für die deutsche Aussprache besser eignet.

Abb. 2.1.4.6 zeigt schließlich einen mit dem SC-01 aufgebauten Sprachsynthesizer. Er findet einschließlich Verstärker und Lautsprecher in einem verhältnismäßig kleinen Gehäuse Platz.

Abb. 2.1.4.6
Kommerzielles Sprachausgabe-Gerät (Gerb)

2.2 Kommunikations-Einrichtungen

2.2.1 Modems

Das Wort „Modem" setzt sich aus „Modulator" und „Demodulator" zusammen. Man bezeichnet damit eine Einrichtung, die es gestattet, Daten über eine niederfrequente Strecke (z. B. das Fernsprechnetz oder eine Funkverbindung) zu übertragen, die ursprünglich für Sprache vorgesehen ist. Demzufolge muß das Datensignal in ein dafür geeignetes Tonsignal umgewandelt werden.

Diesen Vorgang bezeichnet man als Modulation. Grundsätzlich finden in der Modemtechnik drei Modulationsarten Anwendung: die Amplituden-, die Frequenz- und die Phasenmodulation *(Abb. 2.2.1.1)*.

Modulationsarten

Bei der Amplitudenmodulation (AM) wird ein niederfrequentes Tonsignal einfach im Rhythmus der Bits ein- und ausgeschaltet. Empfangsseitig kann man dann an der Unter- oder Überschreitung einer bestimmten Lautstärke (Amplitude) erkennen, ob es sich um ein 0- oder 1-Bit handelt. Leider ist dieses naheliegende Verfahren ziemlich anfällig gegenüber Störsignalen auf der Übertragungs-Strecke. Denn auch ein Störsignal kann (bei Überschreitung einer bestimmten Mindest-Amplitude) ein 1-Bit vortäuschen. Die Amplitudenmodulation wird daher recht selten eingesetzt.

Wesentlich bessere Eigenschaften bietet die Frequenzmodulation (FM). Für die Entscheidung, ob ein 0- oder 1-Bit übertragen wird, ist nicht die Amplitude, sondern nur die Frequenz des Hilfs-Tonsignals ausschlaggebend. Nur sehr starke Störsignale können den empfangsseitigen Demodulator durcheinanderbringen. Die meisten Modems arbeiten heute nach dem FM-Verfahren.

Für höhere Übertragungs-Geschwindigkeiten als etwa 1200 Bit/s eignet sich die Frequenzmodulation leider nicht allzu gut, da der Demodulator eine gewisse Zeit braucht, um eine Frequenzänderung und damit einen Bitwechsel feststellen zu können. Hier kommt die Phasenmodulation zur Anwendung: Das niederfrequente

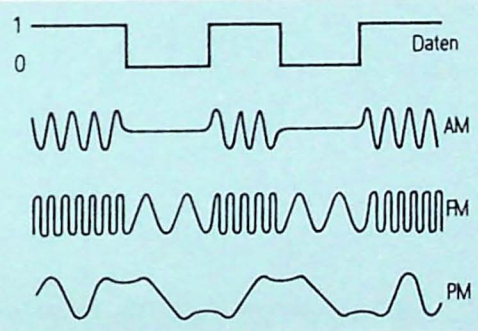


Abb. 2.2.1.1
Die drei Modulationsarten AM, FM und PM

Tonsignal wird nicht in seiner Frequenz, sondern in seiner Phasenlage umgeschaltet, beispielsweise um 180 Grad. Bedingt durch die begrenzte Bandbreite des Übertragungsweges (bei Fernsprechleitungen etwa 0,3...3,5 KHz) wird das resultierende Signal natürlich etwas „verschliffen", wie in Abb. 2.2.1.1 auch gezeigt. Die Störanfälligkeit ist deutlich höher als bei FM.

V.21: 200/300 Bit/s in jeder Richtung

Für FM-Modems hat man sich auf bestimmte einheitliche Frequenzen geeinigt (CCITT-Normen). *Abb. 2.2.1.2* zeigt deren Lage im Spektrum bei Geschwindigkeiten für 200 oder 300 Bit/s (Norm V.21). In jeder Richtung ist ein Kanal, ein Frequenzpaar also, vorhanden, damit jede der beiden Stationen gleichzeitig und unabhängig voneinander senden und empfangen kann (Duplex-Betrieb). Darüber hinaus gibt es noch einen sogenannten Antwortton von 2100 Hz, der zur eigentlichen Datenübertragung aber nicht erforderlich ist, sondern u.a. dem Abschalten von Echosperren bei Satelliten-Ferngesprächen dient. (Geostationäre Satelliten umkreisen die Erde in 36 000 km. Die entstehende Signallaufzeit, hin und zurück, beträgt mindestens etwa 0,3 Sekunden. Um einen hörbaren Echo-Effekt zu unterdrücken, wird die Verbindung immer nur in einer Richtung durchgeschaltet, nämlich von dem Teilnehmer, der gerade spricht, zum anderen. Diese Echosperre würde eine Duplex-Datenübertragung natürlich unmöglich machen.)

Konkret läuft eine Verbindung nach V.21 so ab:

1. Der Anrufer lädt ein spezielles Kommunikations-Programm in seinen Rechner und wählt die Nummer der Gegenstation.
2. Die Gegenstation schaltet beim Klingel-Signal manuell oder automatisch das Modem an die Leitung. Dieses sendet etwa drei bis sechs Sekunden lang den 2100-Hz-Antwortton, um eventuelle Echosperren in der Leitung abzuschalten.
3. Beim Erkennen des Antworttons sendet der Anrufer 980 Hz (Ruhelagepegel = log. 1). Die gerufene Station antwortet mit einem Ruhesignal-Ton von 1650 Hz, sobald die Antwortton-Zeit verstrichen ist.

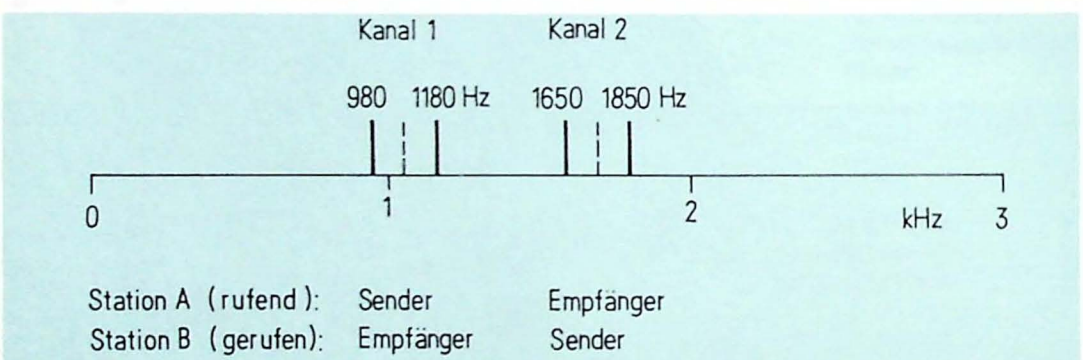


Abb. 2.2.1.2 Lage der Frequenzen bei 300 Bit/s nach CCITT

4. Nun kann die Datenübertragung in beiden Richtungen beginnen. Der Anrufer sendet 980 Hz für ein 1-Bit (Mark) und 1180 Hz für ein 0-Bit (Space); der Angerufene antwortet mit 1650 Hz für 1 und 1850 Hz für 0.
5. Die Verbindung wird entweder durch ein spezielles End-Kommando abgebrochen oder dann, wenn länger als z. B. zwei Sekunden eine Station von der anderen keinen Ton empfängt.

Das Datenformat selbst richtet sich gewöhnlich nach den Empfehlungen der V.24-Schnittstelle, die schon besprochen wurde. Das bedeutet, daß jedem Byte (8 Bits) ein Startbit (0) voran- und ein oder zwei Stopbits (1) nachgestellt werden (Asynchron-Betrieb).

V.23: 600/1200 Bit/s mit Hilfskanal

Bei höheren Übertragungsgeschwindigkeiten werden abweichende Tonfrequenzen verwendet. Die CCITT-Norm V.23 sieht für 600 Bit/s das Spektrum nach *Abb. 2.2.1.3 oben* vor, bei 1200 Bit/s nach *Abb. 2.2.1.3 unten*. In beiden Fällen ist allerdings die angegebene Geschwindigkeit nur in einer Richtung möglich (Datenkanal). In der anderen steht nur ein Hilfskanal mit 75 Bit/s zur Verfügung. So arbeitet z. B. Bildschirmtext (Btx) mit 1200/75 Bit/s. Die Btx-Zentrale sendet mit 1200 Bit/s, und der Teilnehmer kann Steuerkommandos mit 75 Bit/s übermitteln. Dies ist sinnvoll, weil die vom Teilnehmer an die Btx-Zentrale gesendeten Kommandos nur eine geringe Datenmenge darstellen, für die keine hohe Geschwindigkeit nötig ist.

Der Bell-Standard

In den USA werden für Modems überwiegend andere Frequenzen als die oben erwähnten verwendet. Statt nach der CCITT-Empfehlung richtete man sich dort nach einem Vorschlag der Telefongesellschaft Bell.

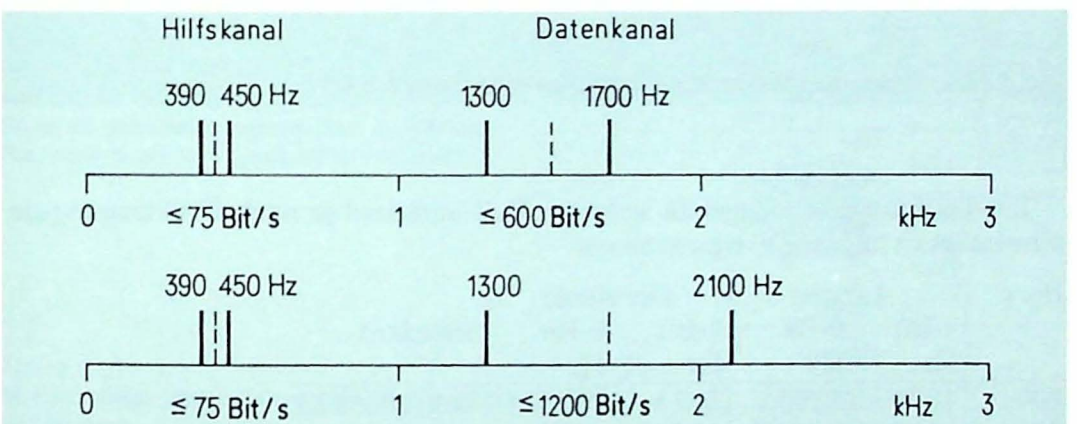


Abb. 2.2.1.3 Lage der Frequenzen bei 600 Bit/s (oben) und 1200 Bit/s (unten). Die Übertragung kann nur in einer Richtung mit dieser Geschwindigkeit erfolgen; in der anderen sind maximal 75 Bit/s vorgesehen

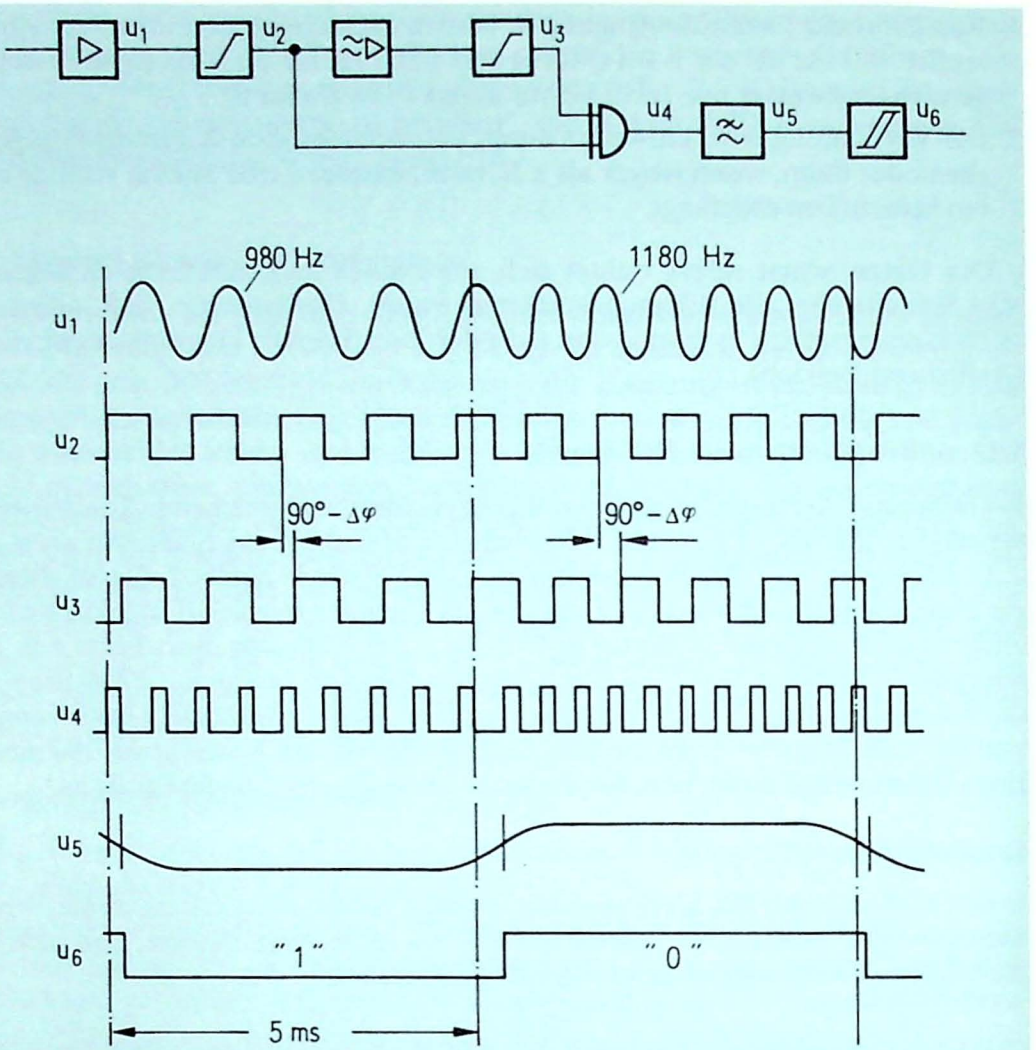


Abb. 2.2.1.4 Prinzip der FM-Demodulation im Siemens-Modem D300S

Tabellarisch zusammengefaßt sieht der Bell-Standard je nach Übertragungsgeschwindigkeit folgende Frequenzen vor:

Bit/s	Anrufer		Gerufener		
	1-Bit Hz	0-Bit Hz	1-Bit Hz	0-Bit Hz	Standard
300	1070	1270	2025	2225	Bell-103/113
600	387	—(AM)	1200	2200	Bell-202

Daraus folgt, daß sich Modems, die für den amerikanischen Markt gebaut wurden, für den innereuropäischen Datenaustausch leider nicht verwenden lassen.

Prinzip und Anschluß eines Modems

Der Modulator-Teil eines FM-Modems besteht einfach aus einem Oszillator, dessen Frequenz im Rhythmus des zu sendenden Datensignals umgeschaltet werden kann. Das läßt sich schaltungstechnisch leicht erreichen, indem man z.B. einen Transistor als elektronischen Schalter zwei unterschiedliche frequenzbestimmende Widerstände eines RC-Oszillators umschalten läßt.

Empfangsseitig muß der Demodulator das FM-Signal in ein Datensignal rückverwandeln. Wie das bei einem industriellen Modem (D-300S von Siemens) geschieht, zeigt *Abb. 2.2.1.4.* Nach einem Empfangsverstärker wird das Signal auf eine konstante Amplitude begrenzt, um überlagerte Störspannungen zu eliminieren; es interessiert ja nur die Frequenz des empfangenen Signals.

Der Signalweg trennt sich nun auf: Einmal wird die Eingangsfrequenz über ein Filter und einen weiteren Begrenzer an einen Eingang eines Exklusiv-Oder-Gatters geleitet und einmal direkt. Da das Filter eine stark frequenzabhängige Phasenverschiebung verursacht, ist die Impulsbreite (u4) am Ausgang des Gatters ebenfalls frequenzabhängig. Ein Tiefpaß glättet die Einzelimpulse zum Datensignal, das durch einen Schmitt-Trigger am Ausgang eine steilflankige Rechteckform erhält. Die Angabe „5 ms" gilt für 200 Bit/s; bei 300 Bit/s dauert ein Bit nur 3,33 ms.

Über die Verbindung des Modems mit dem Fernsprechnetz gibt *Abb. 2.2.1.5* Auskunft. Wenn das Telefon eine Datentaste enthält, kann man mit ihr manuell vom Fernsprech- auf den Modembetrieb umschalten. Andernfalls kann die Umschaltung auch über die Steuerleitung „Modem an Leitung schalten" erfolgen (Pin 20 der V.24-Schnittstelle). Welche Leitungen insgesamt zwischen Rechner und Modem mindestens nötig sind, geht aus *Abb. 2.2.1.6* hervor. Die Bedeutung von Pin 20 der Schnittstelle ist von der internen Brückenlegung des Modems abhängig.

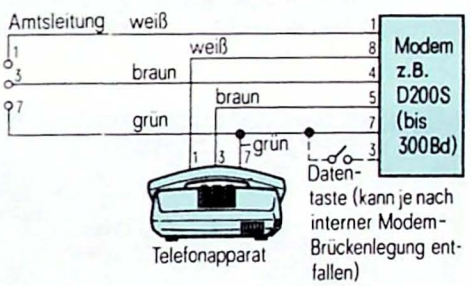


Abb. 2.2.1.5
So ist ein galvanisch gekoppeltes, gemietetes Postmodem mit dem Telefonnetz verbunden

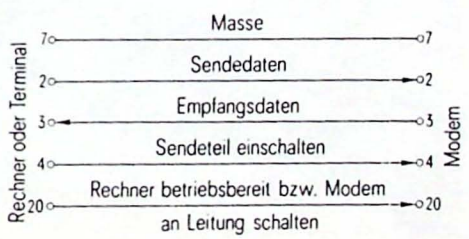


Abb. 2.2.1.6
Minimal nötige Verbindungen zwischen Rechner und Modem

Im Gegensatz zu Akustikkopplern (siehe nächstes Kapitel) können galvanisch gekoppelte Modems sozusagen selbst den Hörer abnehmen (d.h. sich an die Leitung schalten), wenn sie angerufen werden. Dazu besitzen sie einen Schalter „Automatik": Sofern der Rechner über den V.24-Pin 20 meldet, daß er bereit ist und der Schalter am Modem auf Automatik-Betrieb steht, schaltet sich das Modem selbsttätig an die Leitung, sobald ein Anruf eingeht, und sendet etwa 3,5 oder 6,5 Sekunden den Antwortton (s.o.). Über Stift 29 der Schnittstelle wird dem Rechner dann (also erst nach dem Antwortton!) „Sendebereitschaft" gemeldet. Diese Leitung geht wieder in ihre Ruhelage zurück, wenn von der Gegenstation (hier dem Anrufer) kein Tonsignal mehr empfangen wird.

In Deutschland darf man solche galvanisch gekoppelten Modems weder selbst bauen, kaufen noch anschließen. Vielmehr muß man sie von der Post auf der Basis einer monatlichen Gebühr mieten. Diese Gebühr (1985: 80 DM bei 300 Bit/s) ist von der gewünschten Übertragungsgeschwindigkeit abhängig und zu den üblichen Telefongebühren zu addieren.

Die Richtlinien für Modems werden in Deutschland vom Fernmeldetechnischen Zentralamt in Darmstadt aufgestellt. Die Zulassung von neu entwickelten Modemtypen erfolgt in Saarbrücken durch das Zentralamt für Zulassungen im Fernmeldewesen (ZZF). Eine ZZF-Zulassung ist hierzulande auch für die V.24-Schnittstellen von Rechnern erforderlich, an die man galvanisch gekoppelte Modems anschließen möchte. In den USA ist es dagegen zulässig, solche direkt an das Fernsprechnetz gekoppelten Modems selbst über die Telefon-Steckdose anzuschließen, und es ist auf der Rechnerseite auch keine Zulassung für die Schnittstelle erforderlich.

2.2.2 Akustikkoppler

Wenn sich der Anschluß eines gemieteten Postmodems nicht lohnt oder wegen wechselnden Einsatzortes (Hotelzimmer, Telefonzelle usw.) nicht möglich ist, verwendet man sogenannte Akustikkoppler. Sie enthalten zwei Gummi-Manschetten,

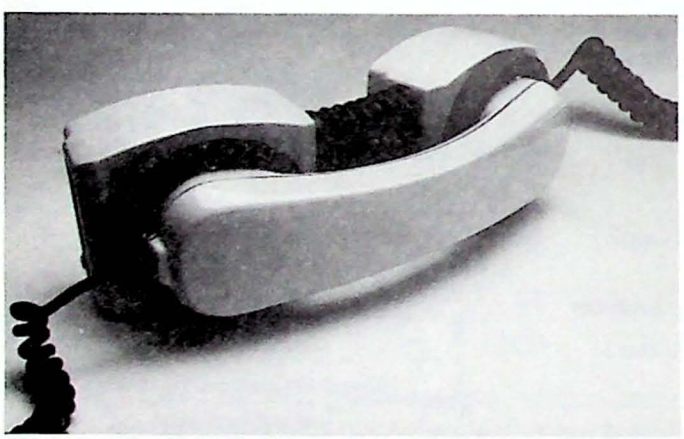


Abb. 2.2.2.1
Akustikkoppler besitzen oft ein flexibles Mittelteil, damit sie mit unterschiedlichen Telefonhörer-Formen betrieben werden können

in die man den Telefonhörer akustisch dicht einlegen kann. Die eine Muschel enthält einen kleinen Lautsprecher für die zu sendenden Daten, die andere ein Mikrofon zum Empfang *(Abb. 2.2.2.1)*.

Wegen der akustischen Ankopplung ist der Einsatz solcher Geräte mit Kompromissen behaftet. Einmal können laute Umgebungsgeräusche Fehler bei der Datenübertragung verursachen. Zum anderen ist wegen der begrenzten Linearität von Phasen- und Frequenzgang bei Mikrofon und Lautsprecher (sowohl im Koppler als auch im Hörer selbst) die erreichbare Übertragungsgeschwindigkeit eingeschränkt.

Akustikkoppler gibt es für 300 Bit/s ab etwa 200 DM, Ausführungen für 1200 Bit/s sind wesentlich teurer. (Im Gegensatz zu galvanisch an das Fernsprechnetz gekoppelten Modems mietet man sie nicht bei der Post, sondern kann sie frei kaufen. Allerdings ist eine Serien-Zulassung durch das ZZF erforderlich.) Bei der Verwendung mit älteren Telefonapparaten, die noch mit Kohlemikrofon-Kapseln ausgerüstet sind, können sich nach einigen Minuten Betrieb Schwierigkeiten ergeben, da die Kohlekörner im Hörer allmählich zusammenbacken. Dann hilft nur, den Hörer kurz zu schütteln. In Schräglage tritt dieses Problem etwas weniger in Erscheinung.

Manche Akustikkoppler eignen sich nur für Verbindungen mit galvanisch gekoppelten Postmodems, da sie nur das Frequenzpaar 980/1180 Hz senden und 1650/1850 Hz empfangen können. Es ist also nicht möglich, zwei derartige Koppler miteinander in Verbindung treten zu lassen.

Anders jene Ausführungen, die einen Umschalter namens „Originate/Answer" oder auch „Call/Answer" besitzen. Dieser Schalter ist auf Call bzw. Originate zu stellen, wenn man selbst jemanden anruft, und auf Answer, wenn man angerufen wird. Natürlich ist allerdings eine automatische Entgegennahme von Anrufen hier nicht möglich, da der Koppler ja den Hörer nicht abnehmen kann.

Bei einem 300-Bit/s-Akustikkoppler nach europäischer CCITT-Norm (V.21) ergeben sich folgende Frequenzen:

	Empfänger		Sender		
	log. 1	log. 0	log. 1	log. 0	
Originate	1650	1850	980	1180	Hz
Answer	980	1180	1650	1850	Hz

Als Modulator und Demodulator verwendet man in Akustikkopplern meist spezielle integrierte Schaltungen. *Abb. 2.2.2.2* zeigt ein Beispiel dafür: Das IC 7910 (Hersteller: Thomson und AMD) eignet sich für alle Frequenzen, die in Europa und in den USA für 300, 600 und 1200 Bit/s üblich sind. Über Steuereingänge zur Betriebsarten-Wahl kann eingestellt werden, ob die amerikanischen Bell- oder die europäischen CCITT-Frequenzen erzeugt und empfangen werden sollen. Als Referenz dient dabei ein einziger Quarz. Der Chip generiert auch alle für die V.24-Schnittstellen nötigen Steuersignale.

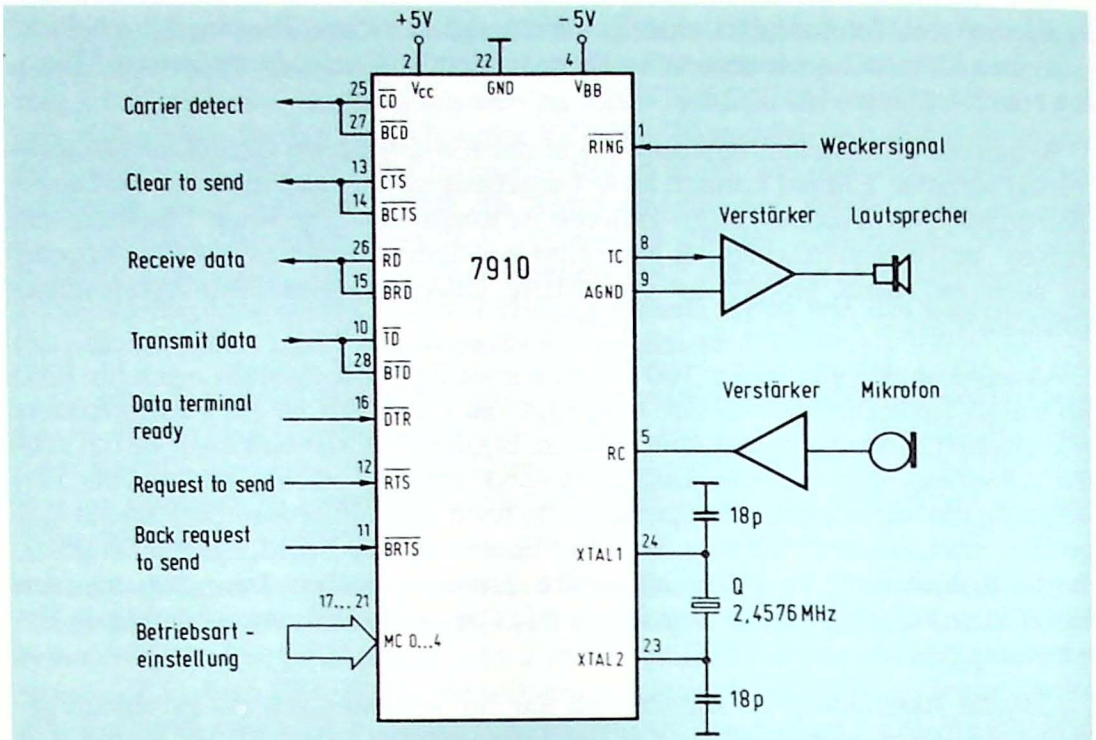


Abb. 2.2.2.2 Beschaltung des Modem-IC 7910/7911 von Thomson/AMD

2.2.3 Der Datex-P-Dienst

Bisher war immer von der Datenfernübertragung (DFÜ) über das öffentliche Fernsprechnetz die Rede. Dieser Verbindungsweg ist zwar naheliegend, kann aber (speziell bei tagsüber geführten Ferngesprächen) recht teuer werden. Ferner ist zur gleichen Zeit immer nur eine Verbindung mit einer einzigen Gegenstation möglich.

Die Deutsche Bundespost führte daher den Datex-P-Dienst ein. Das „P" steht hier für „Paketvermittlung". Damit sind Datenpakete gemeint, die nach einem festgelegten Protokoll (HDLC, CCITT X.25) innerhalb eines eigenen Datennetzes transportiert werden. Da jedes Paket ein Adressierungsfeld enthält, ist es sogar möglich, von einem Datex-P-Anschluß Daten an mehrere Gegenstationen quasi gleichzeitig (nämlich abwechselnd) zu senden. Das Protokoll bietet aber noch andere Vorteile: Bei Übertragungsfehlern muß nicht alles von Anfang an wiederholt werden, sondern nur das fehlerhafte Paket. Und die bei Sender und Empfänger eingestellten Geschwindigkeiten müssen nicht gleich sein: Die Datex-P-Netzknoten sorgen für die notwendige Zwischenspeicherung.

Abb. 2.2.3.1 macht aber deutlich, daß es außer den (rechts sichtbaren) paketorientierten Netzanschlüssen hoher Geschwindigkeit (2400...48000 Bit/s) auch die Möglichkeit gibt, „unintelligente" Terminals und Computer mit Datex-P zu ver-

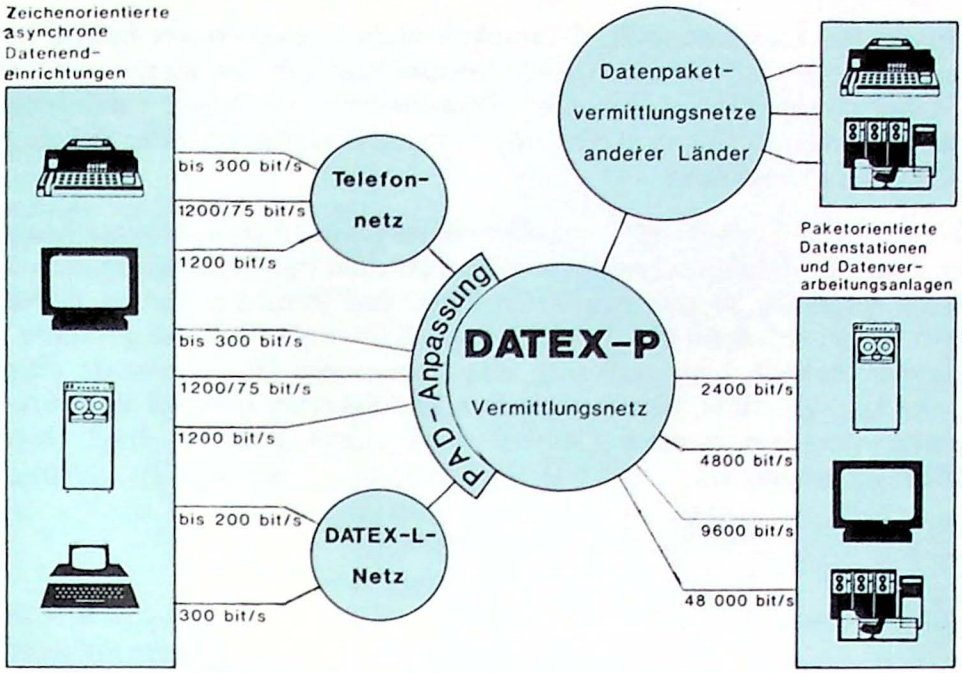


Abb. 2.2.3.1 Zugangsmöglichkeiten zum Datex-P-Netz der Deutschen Bundespost

DATEX-P Vermittlungs-stelle	Ortsnetz-kennzahl (Vorwahl)	Rufnummern für Übertragungsgeschwindigkeiten bis 300 bit/s (D300S)	1200 bit/s (D1200S12)	1200/75 bit/s (D1200S0.)
Augsburg	08 21	3 67 91	3 67 81	3 67 61
Berlin	0 30	24 00 01	24 00 81	24 00 61
Bielefeld	05 21	5 90 11	5 90 21	5 90 41
Bremen	04 21	17 01 31	1 42 91	1 50 77
Dortmund	02 31	5 70 11	5 20 11	5 20 81
Düsseldorf	02 11	32 93 18	32 92 49	32 07 48
Essen	02 01	78 70 51	79 10 21	79 30 03
Frankfurt	0 69	2 02 81	2 02 91	2 02 01
Hamburg	0 40	44 12 31	44 12 61	44 12 81
Hannover	05 11	32 66 51	32 74 81	32 75 91
Karlsruhe	07 21	6 02 41	6 03 81	6 05 81
Köln	02 21	29 11	29 31	29 51
Mannheim	06 21	40 90 85	3 99 41	3 99 51
München	0 89	22 87 30	22 86 30	22 87 58
Nürnberg	09 11	2 05 71	2 05 41	2 05 01
Saarbrücken	06 81	81 00 11	81 00 31	81 00 61
Stuttgart	07 11	29 91 71	29 90 61	29 92 91

Abb. 2.2.3.2 Rufnummern der PAD-Einrichtungen des Datex-P-Netzes

binden, die das komplizierte X.25-Protokoll nicht implementiert haben. In den größeren Städten der Bundesrepublik Deutschland gibt es hierzu sogenannte PAD-Einrichtungen (Packet Assembly/Disassembly): Sie bauen sendeseitig die einzelnen Zeichen zu Datenpaketen zusammen und zerlegen solche Pakete empfangsseitig in Zeichenfolgen.

Abb. 2.2.3.2 nennt die Telefonnummern, unter denen man solche zeichenorientierten Datex-P-Anschlüsse erreichen kann. Hat man die Verbindung zum PAD-Anschluß hergestellt, so muß man einen Punkt und Return eingeben. Der PAD-Rechner meldet sich dann mit „DATEX-P: 44 XX", wobei XX die gewählte Telefonnummer darstellt. Nun muß man eine sogenannte NUI-Nummer eingeben (Network User Identifier, Benutzernummer, im folgenden Beispiel als DAAAAA dargestellt) eingeben, wonach Datex-P nach einem Paßwort fragt (hier als BBBBBB dargestellt):

```
NUI DAAAAA    (Return)
DATEX-P: password
XXXXXX
BBBBBB        (Return)
DATEX-P: teilnehmerkennung DAAAAA aktiv    (bei richtigem Paßwort)
DATEX-P: teilnehmerkennung fehler          (bei falschem Paßwort)
```

Die Teilnehmerkennung (NUI) und das Paßwort erhält man vom Fernmeldeamt. Das Paßwort kann man selbst jedoch jederzeit ändern.

Will man jetzt die Verbindung mit einer Gegenstation herstellen, so gibt man einfach deren Datex-P-Teilnehmerkennung ein. (Allerdings können natürlich nur Datex-P-Hauptanschlüsse — mit galvanisch gekoppelten Modems — angerufen werden.) Stellt man der anzurufenden Nummer REV oder R voraus (reverse charging), so bedeutet das die Anforderung einer Gebühren-Übernahme durch die Gegenstation. Das funktioniert aber natürlich nur, wenn der angerufene Teilnehmer diese Option auch bei der Post angemeldet hat. Mit CUG nnn (Closed User Group) oder C nnn vor der gerufenen Nummer kann man der Gegenstation auch gleich ein eventuell nötiges Paßwort (hier nnn) übermitteln.

Datex-P-Gebühren

Der wesentliche Vorteil von Datex-P gegenüber dem normalen Fernsprechnetz hängt mit der Gebührenstruktur zusammen, die bei Datex-P (zumindest innerhalb von Deutschland) nicht entfernungsabhängig ist. Auf einen Nenner gebracht: Datex-P lohnt sich finanziell insbesondere dann, wenn die getätigten Verbindungen überwiegend über das eigene Fernsprech-Ortsnetz hinausreichen. Bei Fernverbindungen ist der Kostenvorteil von Datex-P eklatant. (Ihr Fernmeldeamt hält eine ausführliche Broschüre mit der genauen Kostenstruktur von Datex-P bereit.)

Die Datex-P-Gebühren setzen sich aus folgenden Elementen zusammen (Stand vom Juli 1985):

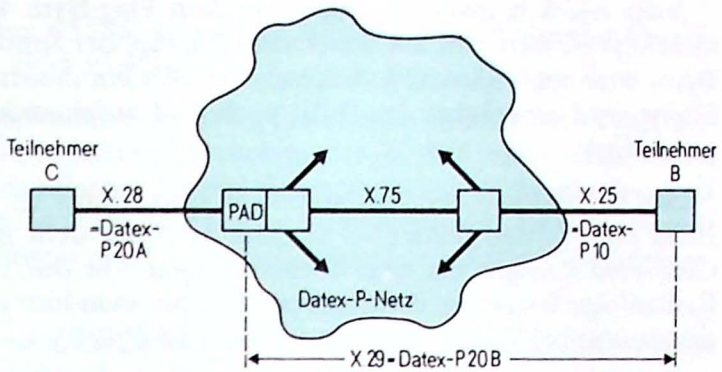


Abb. 2.2.3.3
Datex-P erlaubt Verbindungen von Teilnehmern, die unterschiedliche Protokolle verwenden

Monatliche Grundgebühr:	140 DM (300 Bit/s) bis 2500 DM (48000 Bit/s) (inklusive Mietmodem, Datex-Netz) oder: 15 DM Teilnehmerkennung (ohne Modem, bei Zugang über das Fernsprechnetz)
Ev. Zuschläge hierzu:	z. B. 10 DM zur Gebühren-Übernahme (R-Option)
Verbindungsgebühr:	5 Pf je Verbindung
Gebühr bei Zugang über das Fernsprechnetz:	300 Bit/s: 4 Pf je Verbindung 1200 Bit/s: 5 Pf je Verbindung
Ev. PAD-Gebühr:	6 Pf je Verb. oder 30/40 % der Volumengebühr
Zeitgebühr (national):	1 Pf pro Minute
Volumengebühr (nat.):	0,33 Pf je 64 Zeichen (8—18 Uhr) 0,18 Pf je 64 Zeichen (6—8, 18—22 Uhr) 0,09 Pf je 64 Zeichen (22—6 Uhr) (Mengenstaffel ab 0,2 Mio. Zeichen)

Abb. 2.2.3.3 macht die unterschiedlichen Protokolle innerhalb Datex-P deutlich. X.28 ist die zeichenorientierte Übertragung, X.25 die paketorientierte. Innerhalb des Datex-P-Netzes kommt die CCITT-Norm X.75 zur Anwendung, die eine Erweiterung von X.25 darstellt.

Aufbau eines Datenpakets

Nach der X.25-Norm (HDLC = High-Level Data Link Control) sieht ein Datenpaket (Frame) wie folgt aus:

1. Flag-Byte (Bitmuster 01111110)
2. Adreßfeld (1, 4, 8, 16 oder 24 Byte)
3. Steuerfeld (1 Byte)
4. Datenfeld (128 Byte)
5. Blockprüfzeichen (CRC), 2 Byte
6. Flag-Byte (Bitmuster 01111110)

Jeder Block beginnt und endet mit dem Flag-Byte. Dessen Bitfolge darf sonst nicht vorkommen. Um das sicherzustellen, fügt der Sender innerhalb der sonstigen Bytes nach fünf aufeinanderfolgenden 1-Bits ein zusätzliches 0-Bit ein. Im Empfänger wird umgekehrt das 0-Bit nach fünf aufeinanderfolgenden 1-Bits wieder unterdrückt.

Die Reihenfolge der Bits innerhalb eines Bytes ist unterschiedlich festgelegt: Beim Adreß- und Steuerfeld beginnt man mit dem niederwertigsten Bit, beim CRC-Feld dagegen mit dem höchstwertigen. Für das Datenfeld ist an sich keine Reihenfolge festgelegt, üblicherweise beginnt man hier jedoch mit dem niederwertigsten (wie bei V.24-Schnittstellen bzw. V.28 üblich).

Das Steuerfeld enthält in seinen acht Bits mehrere Informationen. Man unterscheidet S-Frames, die nur Steuerinformationen und kein Datenfeld übertragen, und die „normalen" I-Frames (Informations-Blöcke). Bit 0 des Steuerfeldes ist bei S-Frames 1, bei I-Frames 0. Die Bits 1...3 geben bei I-Frames die Sender-Blocknummer an, bei S-Frames dienen sie der Kommando-Übermittlung. Wenn Bit 4 log. 1 ist, dann wird die Gegenstation zum Senden aufgefordert. Die Bits 5...7 enthalten schließlich die Empfangs-Blocknummer.

Über das CRC-Verfahren war schon im Abschnitt über Floppy-Disk-Aufzeichnung die Rede. Mit zwei CRC-Bytes können Übertragungsfehler mit ei-

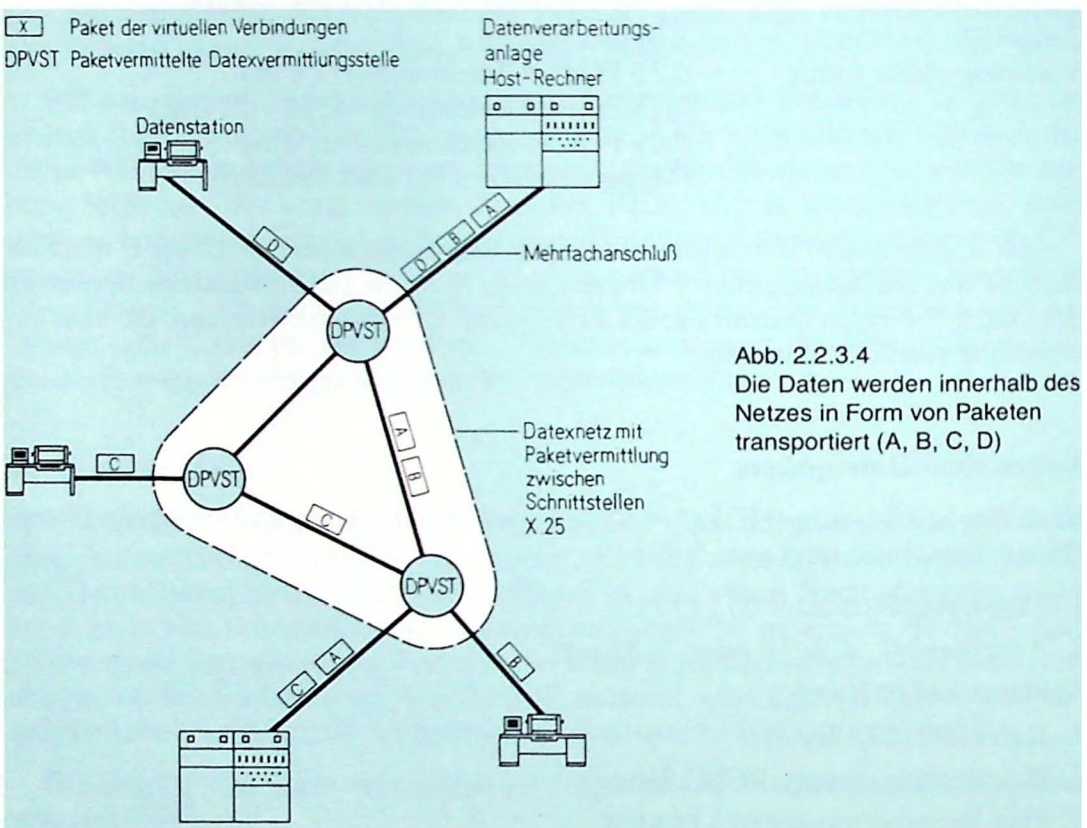


Abb. 2.2.3.4
Die Daten werden innerhalb des Netzes in Form von Paketen transportiert (A, B, C, D)

ner Wahrscheinlichkeit von 65536:1 erkannt werden. Als Generatorpolynom dient dabei die Folge $x^{16} + x^{12} + x^5 + 1$.

Für die Abwicklung dieses doch recht komplizierten Protokolls gibt es spezielle ICs, die sogar die CRC-Generierung übernehmen (z.B. WD 1933). Eine Abart von X.25 hat als AX-25 als „Packet Radio" Eingang in die Amateurfunktechnik gefunden: Hierbei wird ein 24 Byte langes Adreßfeld verwendet, von dem je acht Byte für Absender-, Empfänger- und eventuell Relaisstellen-Rufzeichen vorgesehen sind.

Innerhalb des Datex-P- oder Packet-Radio-Netzes können die einzelnen Pakete je nach Netzbelegung unterschiedliche Wege nehmen. Deshalb ist es auch möglich, daß die (zum Glück numerierten) Pakete in einer anderen Reihenfolge ankommen, als sie abgesendet werden *(Abb. 2.2.3.4)*. Der Empfänger muß sie deshalb vor einer weiteren Verarbeitung zwischenspeichern und wieder in ihre ursprüngliche Reihenfolge bringen.

2.2.4 Datenbanken und Mailbox-Dienste

Mit einem Telefonmodem bzw. einem Datex-P-Anschluß kann man heute eine ganze Reihe von Rechnern anrufen. Für einen Teil davon benötigt man eine Benutzernummer nebst Paßwort, ein anderer Teil ist dagegen, von speziellen Steuerfunktionen abgesehen, öffentlich zugänglich.

Datenbanken enthalten eine Sammlung von Informationen zu einem bestimmten Wissensgebiet. Durch Eingabe von einem oder mehreren, eventuell auch miteinander logisch verknüpften Stichworten ist es möglich, die gewünschten Informationen abzurufen. Die hier aufgeführten Datenbanken sind alle über Euronet-Diane zu erreichen, sozusagen die Erweiterung von Datex-P auf Europa-Ebene. Euronet erreicht man über Datex-P unter der Nummer 023 421 920 100 513; auch ohne Benutzernummer und Paßwort kann man dort Informationen über die angeschlossenen Rechenzentren abrufen.

DIMDI

Deutsches Institut für medizinische Dokumentation und Information, Weißhausstraße 27, Postfach 420580, 5000 Köln 41, Telefon 0221-4724252, Telex 8881364, Euronet 26245221040104 oder 26245221040006. Telex-Anschluß für Suchen: 8882950.

Europäisches Patentamt

European Patents Office (EPO), DG1, BP 5818 Patentlaan 2, 2280 HV-Rijswijk (ZH), Niederlande. Telefon 0031-70-906789, Telex 31651, Euronet 2043030 DEPOIS. Abrufbar ist das gesamte Patentregister.

CMS

Coordinierte Management-Systeme GmbH, Mainzer Landstraße 27, 6000 Frankfurt/Main. Telefon 069-231561, Telex 414742 CMS D, Euronet 2623073. Die Datenbank enthält aktuelle Wirtschaftsdaten.

FIZ-Technik

Fachinformations-Zentrum Technik e.V., Postfach 600547, Ostbahnhofstraße 13, 6000 Frankfurt/Main 1. Telefon 069-4308-255, -227, -225, Telex 4189459 FIZT D, Euronet 2624572474000 1 und 2624572474014 1. Datenbanken über Fachliteratur im gesamten Bereich der Technik.

GID

Gesellschaft für Information und Dokumentation mbH (GID), Sektion für Technik (SFT), Herriotstraße 5, Postfach 710370, 6000 Frankfurt/Main 71. Telefon 069-6687 1, Telex 414351 GIDFM D, Euronet 2624561104000 9DA. Bibliografische Datenbank über Technik.

INFAS

Margaretenstraße 1, 5300 Bonn 2. Telefon 0228-38220, Telex 885575 INFAS D, Euronet 2623074. Datenbank mit statistischen Daten aus Europa.

INKA

Informationssystem Karlsruhe, Fachinformations-Zentrum Energie, Physik, Mathematik, 7514 Eggenstein-Leopoldshafen 2, Telefon 07247-824566, Telex 7826487, Euronet 2624572474000 1 und 2624572474014 1 (also wie FIZ). Datenbank mit wissenschaftlicher Literatur.

Die folgenden US-Datenbanken sind von Europa aus sowohl über das Fernsprechnetz als auch, erheblich kostengünstiger, über Datex-P erreichbar:

The Source

1616 Anderson Road, McLean, VA 22102, Tel. 703-734-7500, USA. Vielbenutzte US-Datenbank mit Kommunikation zwischen den Benutzern, Special-Interest-Groups, Nachrichten aller Art (u.a. von United Press International), ähnlich wie Btx in Deutschland.

CompuServe

5000 Arlington Center Blvd., Columbus, OH 45220, Tel. 614-457-8600, USA. Ähnlich The Source, jedoch mehr kommerzielle Informationen, Nachrichten u.a. von der Washington Post und dem St. Louis Dispatch.

Dow Jones News/Retrieval

P.O.B. 300, Princeton, NJ 08450, Tel. 609-452-2000, USA. Aktuelle Daten und Nachrichten vom Finanzgeschehen, u.a. vom Wall Street Journal, vom Dow Jones News Service und von Barron's.

Für alle diese Datenbanken muß bei der jeweils angegebenen Adresse eine Benutzernummer nebst Paßwort als Zugangs-Berechtigung angefordert werden (natürlich ist das nicht kostenlos!). Darüber hinaus gibt es eine Reihe von Rechnern (meist auf privater Basis), die man auch ohne Benutzernummer anwählen kann. Diese enthalten meist nicht nur eine oder mehrere Datenbank-ähnliche Dateien, sondern auch eine sogenannte Mailbox. Darunter versteht man eine Art elektronischen Briefkasten, in den die Teilnehmer nach Art von Zeitschriften-Kleinanzeigen selbst Informationen, Kontaktwünsche usw. eingeben können. Der erste Dienst dieser Art war der schon Anfang 1983 vom Franzis-Verlag (der auch dieses Buch herausgegeben hat!) eingerichtete „Teledatendienst" TEDAS. Heute gibt es noch einige weitere solche Dienste, deren Parameter sich weitgehend an denen von TEDAS orientieren.

TEDAS

Abb. 2.2.4.1 zeigt, wie sich der Infobox- oder Teledatendienst TEDAS nach Herstellen der Verbindung auf dem Bildschirm des Rechners oder Terminals meldet, nämlich mit einem „Hauptmenü", das acht Auswahlpunkte enthält. Gibt man beispielsweise die Ziffer 2 (und Return) ein, so erscheint das Briefkasten-Untermenü. Nach Eingabe der Ziffer 3 erreicht man schließlich eine weitere Untermenü-Ebene mit einem rubrizierten Briefkasten, der allen Teilnehmern zur Verfügung steht.

Um TEDAS anzurufen, ist es nötig, folgende Übertragungs-Parameter am Rechner oder Terminal einzustellen:

Geschwindigkeit	300 Bit/s (CCITT-Modem)
Datenbits	8, höchstwertiges Bit = 0
Stopbits	1 oder 2
Betriebsart	Vollduplex

Für die Kommunikation sind bei TEDAS folgende ASCII-Steuerzeichen vorgesehen:

Hex-Code	Taste	Zeichen	Wirkung
08	CTRL-H	BS	Ein Zeichen zurück
0A	CTRL-J	LF	Line feed (optional, kann entfallen)
0D	Return	CR	Return, neue Zeile
11	CTRL-Q	XON	Ausgabe nach XOFF weiterlaufen lassen
13	CTRL-S	XOFF	Ausgabe vorübergehend abbrechen
18	CTRL-X	CAN	Ausgabe abbrechen, zurück zum Menü
03	CTRL-C	ETX	Eintrag überspringen
7F	DEL	DEL	Ein Zeichen zurück (wie BS)

```
Bitte Return druecken:

XXX XXX XX  XXX XXX
 X   X    X X X X X
 X   XXX  X X X X XXX
 X   X    X X XXX   X
 X   XXX XX  X X XXX

Sie sind der 63523.Anrufer!
Ausg.stop=CTRL-S;weiter=
CTRL-Q;abbr.=CTRL-X
t = 9
  1 Informationen
  2 Briefkasten
  3 mc-Inhaltsverz.
  4 Softw.-Service-Kat.
  5 Firmenadressen
  6 Bauanl.-Bezugsqu.
  7 Parameter aendern
  8 Ende
Ihre Eingabe: 2
t = 9
  1 >Verlag: Mitt.eingeben
  2 >Mailbox:Mitt.eingeben
  3 Mailbox ansehen
  4 Programme abrufen
  5 Antw.auf Anfragen
  6 Nicht oeffentl.
 99 Hauptmenue
100 Ende
Ihre Eingabe: 3
Welche Rubrik?
t = 9
  1 Tips+Tricks
  2 Biete
  3 Suche
  4 Kontakte
 99 Hauptmenue
100 Ende
Ihre Eingabe: 4
21.1.85 - 63445
... (usw.) ...
```

Abb. 2.2.4.1 So meldet sich der Infobox-Dienst TEDAS auf dem Bildschirm, sobald die Verbindung hergestellt ist. Die Meldung „t=9" vor den Menüs gibt die noch verbleibende Verbindungszeit an

```
0203/787 111 Mercator
0209/271 666 Vollrath
0211/324 517 R-CP/M
0211/3400 71 Data Becker (7E1)
0211/5047865 Epson
02151/801339 KIS
02161/200928 Symic
02202/500 33 Computer-Center
0221/1616284 Saturn
0221/371 076 WDR-Computerclub
02330/74 400 FIS
02331/16 401 Kobrabox Hagen
02373/668 77 Ueding
0251/27 1391 Bit Factory
0251/522 790 Maus
0251/619 054 Mickey
030/414 4068 Gerb-Net
030/66 70 85 Thowo-Box
040/251 2371 MCS
0431/336 038 RBBS Kiel
04348/ 75 13 NCS Kiel
0451/4939 20 MJS
0561/4986 69 DARC
05931/189 48 Compucamp
06128/734 98 Spy-Box
06154/514 33 Decates
06181/488 84 Otis
0621/12 3 02 Spima-Mailbox
06434/ 62 91 CCC-Box
069 /8167 87 Tecos
069/6638 191 Commodore
07082/20 114 Medlog
0711/519 008 Norsak
0721/685 010 MCS Karlsruhe
08 5 44/6 57 Lisa
08121/41 477 Al Capone
0821/52 4035 Resco
0831/69 3 30 GES/Graf
085 31/14 22 Infobox Passau
089/317 1994 Tron
089/39 22 89 Hitech-jr
089/59 64 22 TEDAS 1
089/59 64 65 FSS-Infobox
089/59 84 23 TEDAS 2
089/77 22 13 Typomail
089/79313 32 Phoenix
089/8 31 288 Lauche+Maas
089/8120 338 ACM-Box
089/854 5402 Pro-Box
0911/3300 39 Franken-Box
0911/574 160 Smurf-O-Box
09 120/99 39 J.A.T.
0931/88 4343 Mr.Byte
099 31/68 00 Infoservice
```

Abb. 2.2.4.2
Telefonnummern einiger Infobox-Dienste

Über diesen Mailbox-Dienst hinaus bietet TEDAS mehrere Dateien mit Jahres-Inhaltsverzeichnissen von Computer-Fachzeitschriften, Adressen von Firmen, aktuelle Branchen-Mitteilungen und anderes. Bei den gespeicherten Texten wird von einer Zeilenlänge von 80 Zeichen ausgegangen. In *Abb. 2.2.4.2* sind einige deutsche Mailbox-Dienste aufgeführt (Stand: August 1985, ohne Gewähr). Die Übertragungsparameter orientieren sich bei den meisten am Vorreiter TEDAS.

Bildschirmtext

1984 wurde von der Deutschen Bundespost der Bildschirmtext-Dienst (Btx) bundesweit eingeführt. Die Post stellt hierfür den zentralen Rechner (in Ulm) zur Verfügung, in dem Informationsanbieter Texte und Grafiken in Form ganzer Bild-

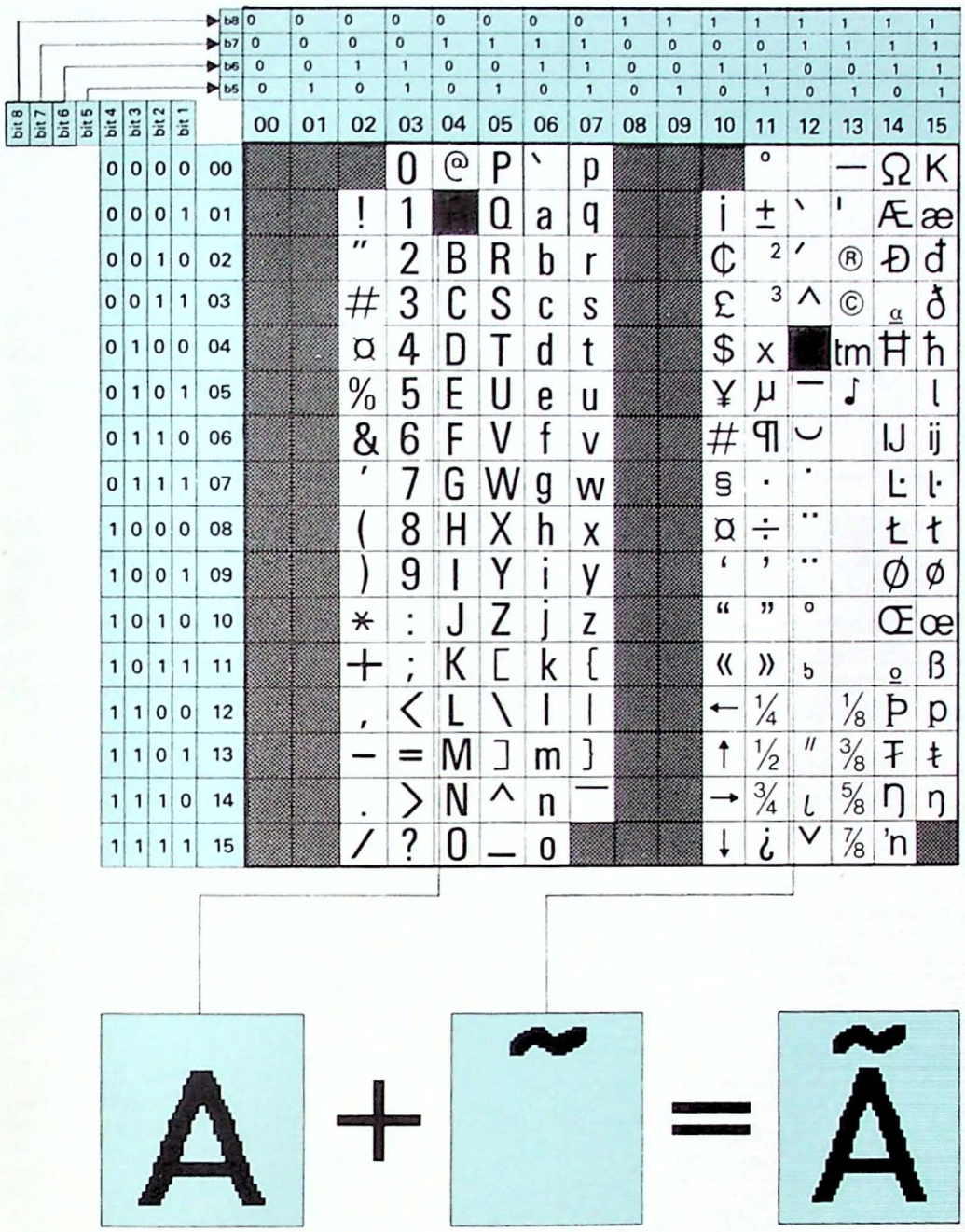


Abb. 2.2.4.3 Der Bildschirmtext-Zeichenaufbau läßt Kombinationen zweier Zeichen zu

schirmseiten speichern können. Die Anbieter haben aber auch die Möglichkeit, den Postrechner nur zur Informations-Weiterleitung zu verwenden und die Daten in einem hauseigenen Rechner zu speichern (Rechnerverbund).

Die Möglichkeit, Grafiken zu übertragen, wird bei Btx mit der sogenannten DRCS-Technik erreicht (Dynamically Redefinable Character Set, dynamisch veränderbarer Zeichensatz). Das bedeutet, daß außer dem in *Abb. 2.2.4.3* dargestellten alphanumerischen Grund-Zeichensatz sowie einigen hier nicht gezeichneten Blockgrafik-Symbolen auch die Möglichkeit besteht, den Zeichengenerator beim Teilnehmer fernzuladen.

Btx arbeitet mit einer Übertragungsgeschwindigkeit von 1200 Bit/s vom Rechner zum Teilnehmer und 75 Bit/s vom Teilnehmer zum Postrechner, da die vom Teilnehmer eingegebene Datenmenge normalerweise wesentlich geringer ist als die von ihm empfangene. Eine Bildschirmseite besteht aus 20 oder 24 Zeilen mit je 40 Zeichen. Jedes Zeichen wird aus 12 × 12 (bei 20 Zeilen) oder 12 × 10 Punkten (bei 24 Zeilen) gebildet. Insgesamt stehen 335 feste Schriftzeichen, 64 Blockgrafik-Zeichen und 84 bzw. 94 fernladbare Zeichen (DRCS) zur Verfügung. Es können acht Farben bei voller, weitere acht bei reduzierter Intensität oder insgesamt 4096 Farbtöne durch Mischen aus je 16 Abstufungen von Rot, Grün und Blau dargestellt werden. Ferner ist es möglich, die Zeichen zeilenweise auf doppelte Höhe und/oder doppelte Breite zu vergrößern oder blinken zu lassen.

Diese CEPT-Norm ist zwar sehr komfortabel; nützt man aber z. B. die Möglichkeit aus, alle 94 möglichen DRCS-Codes fernzuladen, so kann der Aufbau einer Grafikseite trotz 1200 Bit/s schon einmal eine Minute dauern. Die Übertragung selbst erfolgt asynchron mit 8-Bit-Zeichen und CRC-Fehlerprüfung. Die komplette CEPT-Norm hier zu beschreiben, wäre kaum sinnvoll; sie ist durch Überweisen von 20 DM jedoch vom Postgiroamt Frankfurt, Konto 5904-609, unter der Bestellnummer 157 D2E zu beziehen.

Der Btx-Teilnehmer benötigt ein Farbfernsehgerät mit eingebautem oder externem Btx-Decoder *(Abb. 2.2.4.4)*. Die Aufgaben dieses Decoders können prinzi-

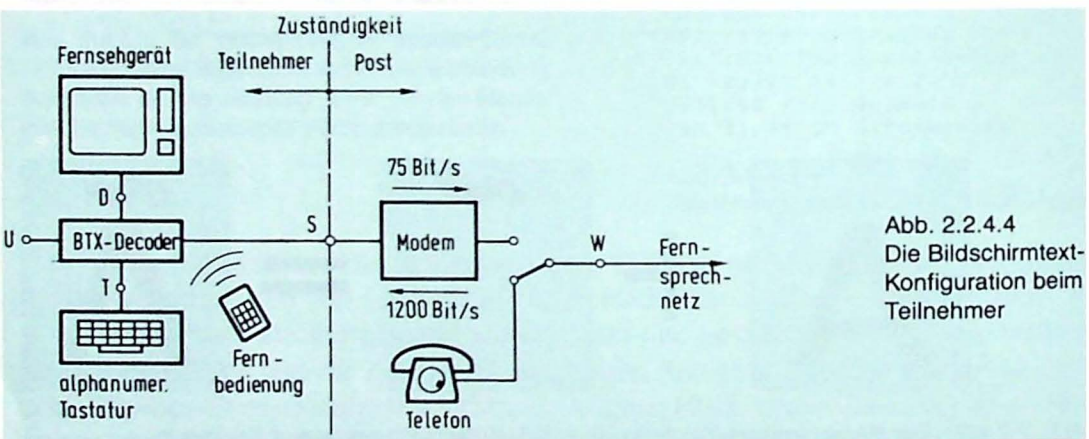


Abb. 2.2.4.4
Die Bildschirmtext-Konfiguration beim Teilnehmer

piell auch von einem Computer übernommen werden, sofern dieser mit einem dafür geeigneten Video-Interface ausgestattet ist (z.B. mit dem Valvo-Eurom-IC). Der Decoder wird über das Modem (von der Post die Btx-Anschlußbox genannt) mit dem Fernsprechnetz verbunden.

Der Telebox-Dienst

Da einerseits Btx die Kommunikation der Teilnehmer untereinander kaum fördert, andererseits aber gerade dafür ein erheblicher Bedarf festzustellen ist, führte die Deutsche Bundespost im Herbst 1984 den Telebox-Dienst ein *(Abb. 2.2.4.5)*. Er ist allerdings wie Datex-P nur durch kostenpflichtige Beantragung einer Teilnehmerkennung benutzbar. Zu erreichen ist der in Mannheim installierte Telebox-Rechner über folgende Anschlüsse:

Telefon, 300 Bit/s	06 21/41 30 91 (vollduplex)
Telefon, 1200/1200 Bit/s	06 21/41 20 71 (halbduplex)
Datex-P (alle Geschw.)	45621040000 und 45621090000

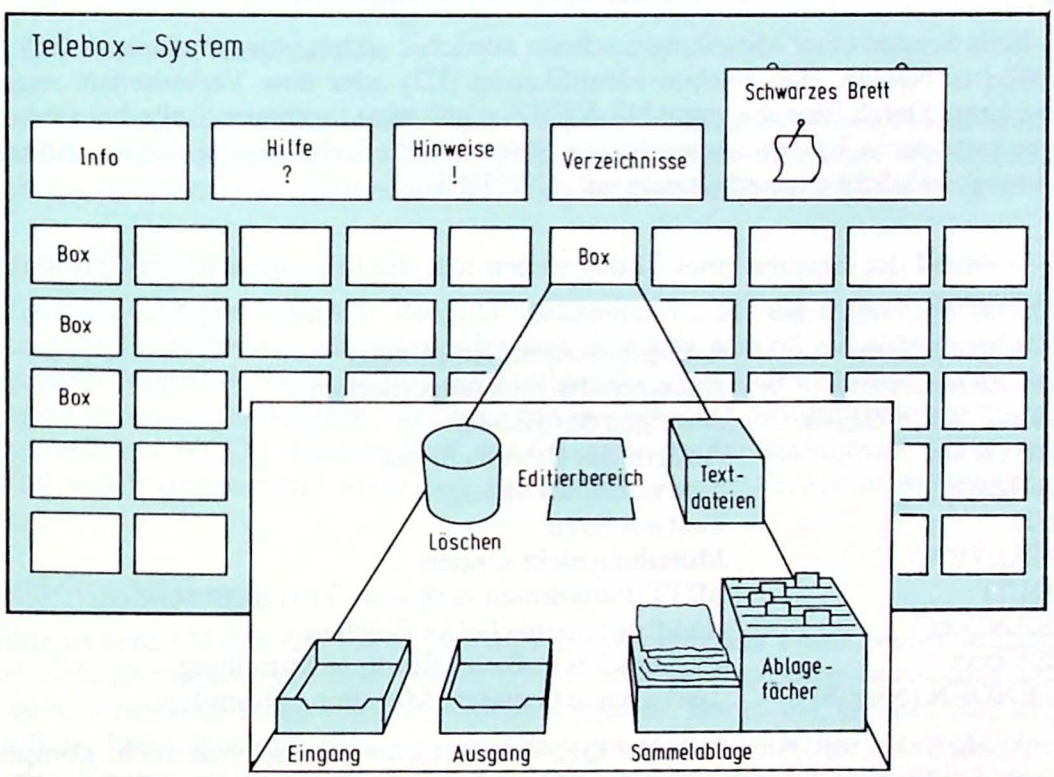


Abb. 2.2.4.5 So kann man sich das Telebox-System der Deutschen Bundespost bildhaft vorstellen

Im Gegensatz zu den meisten anderen Mailbox-Diensten arbeitet Telebox nicht mit einer Menü-Benutzerführung, sondern mit bestimmten Befehlsworten. Die wichtigsten sind:

ENDE	Verlassen des Telebox-Systems
INFO BEFEHLE	Übersicht über alle Befehle
INFO MITT	Informationen über die Mitteilungs-Funktionen
INFO INFO	Inf. über alle gespeicherten Infos
INFO PASSWT	Inf. über Paßwort-Änderung
MITT LESEN	Eingegangene Mitteilungen ansehen
MITT SENDEN	Mitteilungen an andere Telebox-Teilnehmer senden
MITT LESEN UNGEL	Bisher noch ungelesene Mitteilungen lesen
MITT LESEN LALL	Alles lesen und löschen

Nach dem Lesen einer Mitteilung erhält man die Frage „Weitere Verwendung?". Hier sind u.a. folgende Antworten möglich:

LOESCHEN	Gelesene Mitteilung löschen
ANTWORTEN	Mitteilung beantworten
NOCHMALS	Text nochmals anzeigen
Return-Taste	Nächste Mitteilung anzeigen
QUIT	Verlassen der MITT-Funktionen

Beim Senden einer Mitteilung erscheint zunächst „An:", worauf man einen persönlichen Namen, eine Telebox-Identifikation (ID) oder eine Verteilerliste angeben kann. Durch Eingabe von ANZ VERZ erhält man an dieser Stelle bei Bedarf eine Liste der in Telebox angemeldeten Namen. Verteilerlisten und weitere Adressierungs-Möglichkeiten erhält man mit ANZ REF.

Während der Eingabe eines Textes stehen u.a. die folgenden Kommandos zur Verfügung:

.AN Name Name	Anfügen weiterer Empfänger
.ANZEIGEN AN	Anzeigen der Empfängernamen
.ANZEIGEN BETR	Anzeigen des Betreffs
.BETR xxx	Ändern des Betreffs in xxx
.ANZEIGEN	Text nochmals anzeigen
.ED	Text editieren
.HALTEN	Mitteilung nicht senden
.QUIT	MITT-Funktionen verlassen, Text nicht senden
.RAND xx	Randbegrenzung bei xx Zeichen
.PA xxxx	Zusätzliches Paßwort für diese Mitteilung
.SENDEN (oder .S)	Texteingabe beenden, Mitteilung absenden

Ähnlich wie bei Datex-P ist die Gebührenstruktur von Telebox recht kompliziert. Ausführliche Informationen hierüber erhalten Sie beim nächsten Fernmeldeamt.

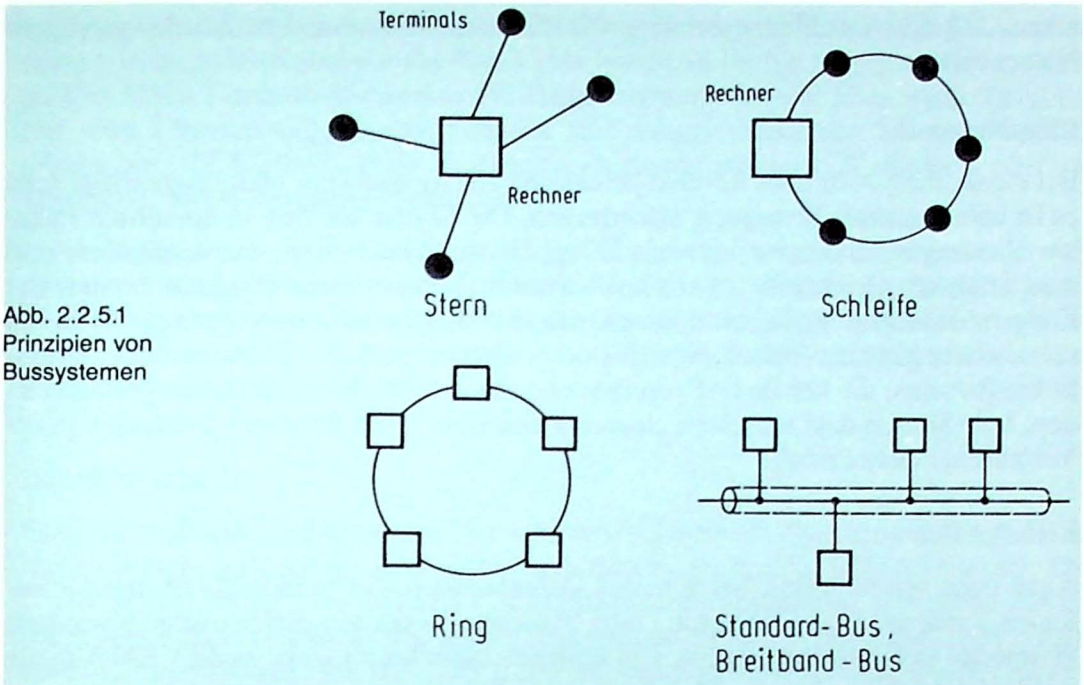


Abb. 2.2.5.1
Prinzipien von
Bussystemen

2.2.5 Lokale Netzwerke

Sind innerhalb eines Gebäudekomplexes mehrere Rechner, Terminals und andere Geräte untereinander verbunden, so spricht man von einem lokalen Netzwerk. Ein solches Netzwerk kann unterschiedlich aufgebaut sein *(Abb. 2.2.5.1)*.

Stern-Struktur

Wenn ein einziger, zentraler Rechner vorhanden ist, der mit mehreren Ein- und Ausgabe-Einheiten verbunden werden soll, so findet man oft ein sternförmig aufgebautes Netzwerk. Hierbei muß der Rechner für jedes angeschlossene Gerät eine eigene Schnittstelle besitzen, was natürlich nur bei einer geringen Zahl von Terminals effizient ist. Der Hauptvorteil des Stern-Netzes ist, daß normale V.24- oder TTY-Schnittstellen ohne besondere Protokoll-Steuerung verwendet werden können.

Schleifen-Struktur

Eine weitere Möglichkeit zur Realisation eines Netzwerkes wurde bereits im Kapitel „Schnittstellen" unter dem Begriff IL-Bus bzw. Interface Loop besprochen. Die Daten werden in einer solchen Schleifen-Struktur immer nur in einer Richtung (z.B. im Uhrzeigersinn) transportiert. Die Steuerung und Abfrage der angeschlossenen Geräte erfolgt durch einen zentralen Controller bzw. Rechner. Jede Station gibt die Daten an die nächste weiter und fügt ggf. eigene an. Nachteilig ist hier vor

allem, daß der Ausfall eines einzigen Geräts zum Zusammenbruch des gesamten Netzes führt.

Ring-Netzwerke

Bei einem Ring-Netzwerk können prinzipiell alle Teilnehmer gleichberechtigt sein, es ist keine zentrale Steuerung erforderlich. Die Daten werden in einzelnen Paketen übertragen, und zwar immer z.B. im Uhrzeigersinn. Um zu vermeiden, daß zwei Stationen gleichzeitig zu senden beginnen, kann man im Ring ein bestimmtes Freigabe-Bitmuster zirkulieren lassen, das jene Station sendet, die als letzte Daten verarbeitete (Control-Token, Newell Loop). Oder eine Leitstation sendet Zeitmultiplex-Rahmen, die bei Bedarf von den einzelnen Stationen mit Daten gefüllt werden. Jede Station darf nur einen „leeren Zeitschlitz" zum Senden verwenden (Slot-Verfahren, Pierce Loop).

Serieller Bus

Auch beim seriellen Bus, bei dem der Datentransport in beiden Richtungen stattfinden kann, muß eine Kollision (zwei Stationen beginnen gleichzeitig zu senden) vermieden bzw. erkannt werden. Ein übliches Verfahren hierzu heißt CSMA (Carrier Sense Multiple Access, Mehrfach-Zugriff durch Träger-Erkennung) oder einfach CD (Collision Detection, Kollisions-Erkennung). „Träger" steht hier für die bei Modems verwendete Modulations-Hilfsfrequenz, die Erkennung eines Ruhepegels auf einem rein digitalen Bus ohne Modems funktioniert aber ebenso. Die Station, die senden will, prüft zunächst, ob der Bus belegt ist. Kommt es nun zu einer Kollision, weil eine andere Station genau gleichzeitig auch sendet, so wiederholen beide Stationen ihren Zugriffsversuch nach einer durch einen Zufallsgenerator gelieferten Verzögerungszeit. Mit großer Wahrscheinlichkeit wird so beim nächsten Versuch eine Kollision vermieden.

Ein solches Netzwerk wurde mittlerweile durch die IEEE-802- und ECMA-TC24TGLN-Empfehlungen zum Standard erhoben. Er eignet sich für Entfernungen bis etwa 3 km. Allerdings muß mit steigender Leitungslänge die Pause zwischen zwei Datenpaketen erhöht werden, da andernfalls das CSMA-Verfahren wegen der nicht mehr zu vernachlässigenden Laufzeiten eine Kollision u.U. zu spät erkennt.

Breitband-Netzwerke

Dieses Standard-Bus-Netzwerk, auch Basisband-Netz genannt, hat natürlich einen kleinen Nachteil: Daten können auf dem Kabel nicht gleichzeitig in beiden Richtungen transportiert werden (Vollduplex), sondern nur halbduplex, d.h. abwechselnd durch eine geeignete Protokollsteuerung, z.B. HDLC bzw. X.25 (siehe Datex-P). Wo Vollduplex-Betrieb erforderlich ist, kann man ein Breitband-Netz einsetzen, das prinzipiell wie die Duplex-Übertragung mit Modems über eine Telefonleitung funktioniert: Die Daten werden nicht direkt digital auf die Leitung gegeben, sondern auf eine Hilfsfrequenz aufmoduliert. Für jede Übertragungsrichtung

ist eine eigene Frequenz vorhanden. Allerdings beträgt die Bandbreite bei Verwendung von Koaxial-Kabeln nicht nur 3 kHz wie beim Fernsprechnetz, sondern einige 100 MHz. Deshalb kann man die Trägerfrequenzen recht hoch wählen und damit eine Übertragungsgeschwindigkeit von einigen MBit pro Sekunde erzielen (Wang Net: 12 MBit/s). Eine Kollisionserkennung ist hier z.B. mit dem oben erwähnten CSMA-Verfahren machbar.

Ist dagegen eine so hohe Geschwindigkeit nicht erforderlich, so kann man statt dessen auch mehr als nur zwei Kanäle zur Verfügung stellen. Im Extremfall können über das gemeinsame Koaxial-Kabel alle Geräte unabhängig voneinander mit jeweils einem anderen kommunizieren, ohne sich gegenseitig zu behindern, so daß keine Kollision auftreten kann.

Das SP-Netzwerk

Ein Beispiel für einen digitalen Bus ist das SP-Netzwerk (Synchron-Polling). Es ist recht einfach realisierbar und eignet sich besonders gut für Mikrocomputer. Die Übertragung erfolgt wie bei einer V.24-Schnittstelle byteweise asynchron mit einem Start- und zwei Stopbits. Jedes angeschlossene Gerät (maximal 255) besitzt eine 1-Byte-Adresse (hex 00...FE), mit der jeder gesendete Datenblock beginnt. Die Ankopplung an die gemeinsame Leitung erfolgt sendeseitig durch einen Ausgang mit offenem Kollektor (wired OR, *Abb. 2.2.5.2*). Im Ruhezustand ist die Leitung auf High-Pegel; sie geht auf Low, wenn sie von einem oder mehreren Geräten nach Masse „gezogen" wird.

Das Grundprinzip ist nun: Wenn ein Gerät sendet, darf es das nächste Bit nur dann auf die Leitung geben, wenn der tatsächliche Leitungspegel mit dem aktuellen (zuletzt gesendeten) übereinstimmt. Da alle Geräte unterschiedliche Adressen haben, bleibt selbst dann immer nur genau ein Gerät übrig (nämlich das mit der niedrigsten Adresse), wenn alle gleichzeitig zu senden beginnen. Das Byte hex FF darf hier nicht als Adresse verwendet werden, da es als Busfreigabe-Kommando dient, um jene Geräte zum Senden ihrer Adresse aufzufordern, die Daten abgeben möchten.

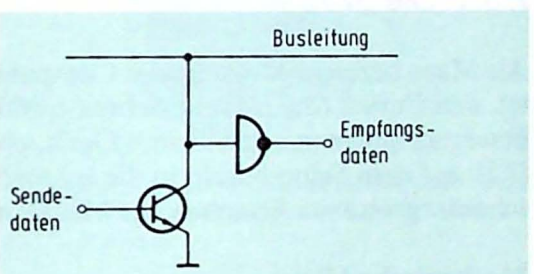


Abb. 2.2.5.2
Interface-Schaltung für den SP-Bus

Herstellerspezifische Netzwerke

Wegen des aufwendigen Protokolls und der bezüglich der Geschwindigkeit sehr hohen Hardware-Anforderungen hat sich der Ethernet-Standard bei Mikrocomputern bisher kaum durchsetzen können. Es existieren daher zahlreiche hersteller-

und typen-spezifische Netzwerke, die den Anschluß von Fremdprodukten leider nicht erlauben (von den Herstellerfirmen aus gesehen ist das offenbar meist sogar eine angenehme Begleiterscheinung). Die wichtigsten solcher Netzwerke sind in der *Tabelle* aufgeführt.

Tabelle: Marktübliche lokale Netzwerke

Bezeichnung	Anbieter	Verfahren	Max. Stationen	Geschwindigkeit
Cluster One	Nestar	Basisband-CSMA	64	250 kBit/s
DEC-Dataway	DEC	verdrillte Ltg.	31	56 kBit/s
Ethernet	Intel u.a.	Basisband-CSMA	1024	10 MBit/s
Econet	Acorn	Basisband-CSMA	255	250 kBit/s
HP-IB	HP u.a.	IEC-Bus	15	1 MBit/s
HP-IL	HP	verdrillte Ltg.	31/961	5 kBit/s
Mars-Net	Syntech	verdrillte Ltg.	255	800 kBit/s
Net/One	Ungermann	Basisband-CSMA	250	10 MBit/s
Omnilink	Northern T.	Breitband/Token	9	40 kBit/s
Omninet	Corvus	verdrillte Ltg.	64	1 MBit/s
Planet	Racal-Milgo	Ring/Token	500	10 MBit/s
Primenet	Prime	Ring/Token	200	8 MBit/s
Saturn	Siemens	verdr. Ltg./Stern	1024	19200 Bit/s
Wangnet	Wang	Breitband-CSMA	65535	12 MBit/s
Z-Net	Zilog	Basisband-CSMA	255	800 kBit/s

2.3 Spezielle Eingabeverfahren

2.3.1 Die „Maus"

Als Maus bezeichnet man in der Computertechnik ein Eingabegerät, das es gestattet, den Cursor (die aktuelle Schreibposition auf dem Bildschirm) hin- und herzubewegen, indem man ein kleines Gerät, eben die Maus, auf einer flachen Unterlage (z.B. auf dem Schreibtisch) in die entsprechende Richtung verschiebt. Man unterscheidet grob zwei Arten solcher Mäuse: mechanische und optische.

Mechanische Mäuse

Die mechanisch aufgebauten Mäuse *(Abb. 2.3.1.1)* arbeiten mit einer Rollkugel. Bewegt man eine solche Maus auf dem Schreibtisch, so folgt die beweglich aufgehängte Kugel durch Rollreibung der Oberfläche in beiden Freiheitsgraden (X- und Y-Richtung). Für X und Y sind nun zwei Reibscheiben vorhanden, die wiederum

Abb. 2.3.1.1 Mechanische Maus, hier an einem Apple-IIe. Sie ermöglicht eine schnelle, bequeme Cursor-Positionierung auf dem Bildschirm

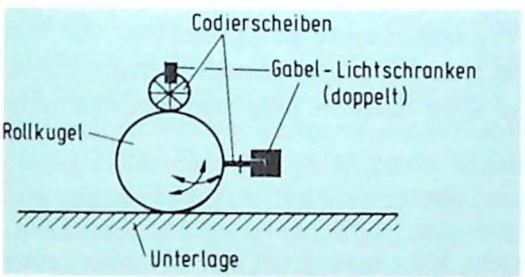


Abb. 2.3.1.2 Funktionsweise einer mechanischen „Maus". Läßt man die Unterlage weg und dreht das Ganze um, hat man eine sogenannte Rollkugel vor sich

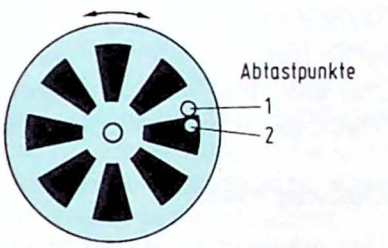


Abb. 2.3.1.3 Mit zwei optischen Sensoren läßt sich die Drehrichtung einer Scheibe erkennen

der Kugelbewegung folgen *(Abb. 2.3.1.2)*. Die Scheiben besitzen helle und dunkle Sektoren *(Abb. 2.3.1.3)*. Auf diese Weise ist es mit Hilfe zweier Fototransistoren oder -dioden je Scheibe möglich, die Umdrehungsrichtung festzustellen. Sie liefern nämlich eine Ausgangsspannung, die log. 1 (z.B. 5 V) ist, wenn der jeweilige Tran-

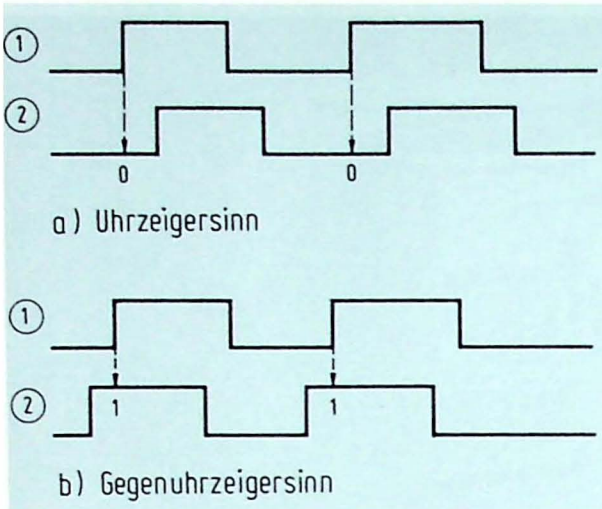


Abb. 2.3.1.4
Prinzip der Drehrichtungs-Erkennung durch Vergleichen der Phasenlage der Ausgangsspannungen beider Sensoren

sistor über einem dunklen Sektor steht, und 0, wenn er über einem hellen steht. (Die Beleuchtung kann z. B. mit einer LED erfolgen.)

In *Abb. 2.3.1.4* ist zu erkennen, wie man daraus die Bewegungsrichtung einer Scheibe ableiten kann. Bei einer 0/1-Flanke des einen Signals prüft man den logischen Zustand des anderen. Ist dieser Null, so dreht sich die Scheibe im Uhrzeigersinn; ist er 1, so handelt es sich um eine Drehung gegen den Uhrzeigersinn.

In einer solchen Maus ist gewöhnlich gleich ein kleiner Einchip-Mikrocomputer eingebaut, der aus den insgesamt vier Signalen (je zwei für jede Scheibe) die Information gewinnt, wie schnell und in welche Richtung die Maus über die Tischoberfläche bewegt wird. Diese Information setzt er dann gleich in geeignete Steuerzeichen zur Bewegung des Cursors auf dem Bildschirm um, so daß eine Verbindung über eine normale Tastatur-Schnittstelle möglich ist.

Eine Abart dieses Prinzips stellt übrigens die sogenannte Rollkugel dar, die fest im Rechner montiert ist und mit der Hand in jede Richtung gerollt werden kann. Das kann man sich so vorstellen wie eine auf dem Rücken liegende mechanische Maus, das Abtast-Prinzip ist das gleiche.

Optische Mäuse

Um die teure Rollkugel-Mechanik einzusparen, kam man früh auf die Idee, eine Maus auf rein optischer Basis zu bauen. Eine denkbare Lösung ist, auf der Tischoberfläche ein sich wiederholendes Muster zu haben, auf der ein herkömmlicher Strichcode-Leser (mit nur einem einzigen optischen Abtast-Element) bewegt wird. Ein solches Muster zeigt *Abb. 2.3.1.5.* Je nachdem, in welche Richtung man diese Maus bewegt, liefert der eingebaute Fototransistor unterschiedliche Impulsformen *(Abb. 2.3.1.6).*

Eine so einfach aufgebaute optische Maus ist aber mit mehreren Mängeln behaftet. Die Breite der dünnen Striche muß mindestens etwa 0,3 mm betragen, weil das

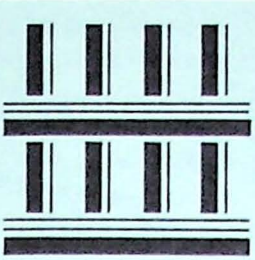


Abb. 2.3.1.5 Durch geeignetes Bedrucken einer Unterlage lassen sich auch optisch arbeitende „Mäuse" realisieren

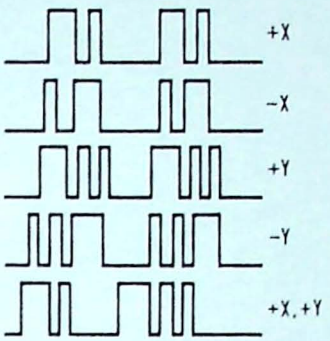
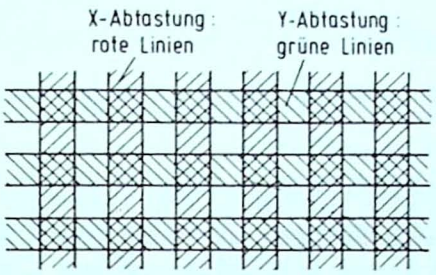


Abb. 2.3.1.6 Impulsdiagramme beim Überfahren des Musters in Abb. 2.3.1.5 für fünf Richtungen



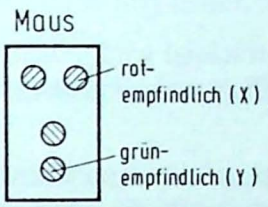


Abb. 2.3.1.7 Arbeitsweise einer optischen Maus mit mehrfarbiger Abtastung

die Auflösung herkömmlicher Strichcodeleser ist. Rechnet man für einen breiten Strich 1 mm, so kann ein Feldelement nicht kleiner als etwa 3 mm im Quadrat gemacht werden. Da ferner der Computer zu Beginn und Ende einer Bewegung, wenn die Geschwindigkeit sich stark ändert, schmale und breite Striche nicht richtig unterscheiden kann, ist die erreichbare Positionier-Genauigkeit bestenfalls etwa 6...10 mm. Für das gezeichnete Muster gilt zudem die Einschränkung, daß u. U. eine genau horizontale Bewegung (entlang einer Linie) nicht erkannt wird. Zwar sind andere Muster denkbar, eine optimale Lösung ist so aber nicht erreichbar.

Es liegt nun nahe, das Prinzip der optischen Abtastung mit zwei Sensoren für jede Richtung aus Abb. 2.3.1.4 zu übernehmen. Um X- und Y-Richtung getrennt auswerten zu können, wählt man die Farbempfindlichkeit der Sensoren für X und Y unterschiedlich, z. B. rot und grün *(Abb. 2.3.1.7)*. (Prinzipiell sind auch andere Farben denkbar, u. U. sogar im Infrarot-Spektrum!) Insgesamt sind hier wie bei der Rollkugel-Maus vier Fototransistoren nötig. Die Auswertung kann wieder durch einen eingebauten Einchip-Mikrocomputer erfolgen, der einen problemlosen Anschluß an den Rechner erlaubt. *Abb. 2.3.1.8* zeigt, daß es sogar möglich ist, die Unterlage mit einer dritten Farbe zu bedrucken, ohne daß die Auswertung der beiden

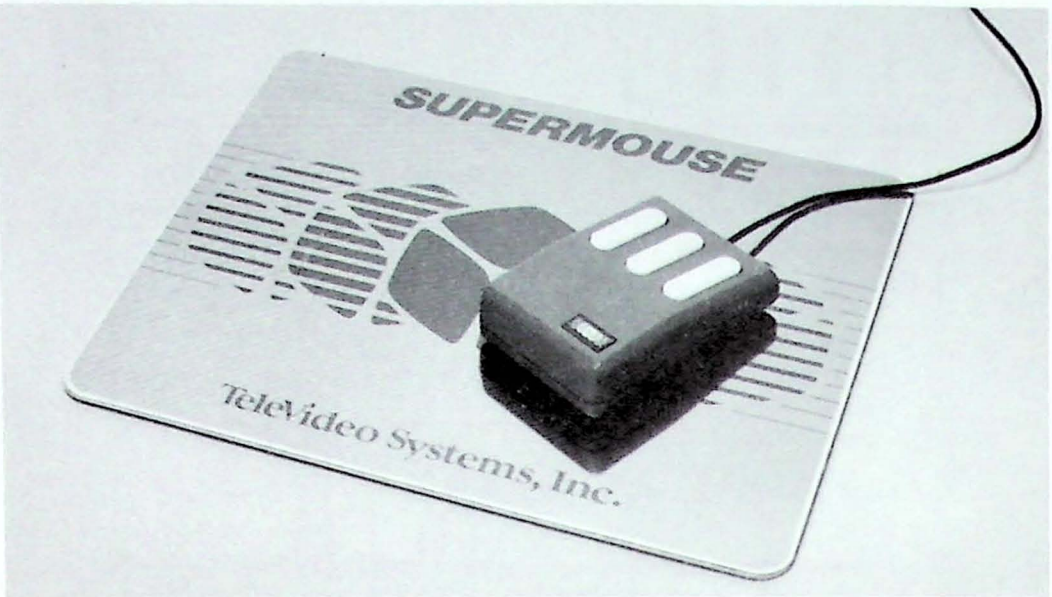


Abb. 2.3.1.8 Diese optische Maus arbeitet mit einer Unterlage, die mit grünen und roten Linien bedruckt ist

anderen beeinträchtigt wird. Auch Fremdlicht kann sich nicht auswirken, weil die Maus flach aufliegt und eingebaute LEDs für die optimale Beleuchtung der Unterlage sorgen.

Der Nachteil solcher optischen „Mäuse" ist natürlich, daß sie nicht auf einer normalen Tischoberfläche, sondern nur zusammen mit der speziellen Unterlage funktionieren, die zudem (je nach Qualität) kratzempfindlich ist. Auf der anderen Seite sind sie aber sehr zuverlässig, da sie keinerlei bewegte Teile enthalten.

2.3.2 Joysticks

Als Joystick bezeichnet man einen vorwiegend bei Computer-Spielen verwendeten Steuerknüppel, mit dem sich Bewegungsabläufe auf dem Bildschirm steuern lassen. Etwas stabiler und präziser aufgebaute Joysticks findet man aber auch bei professionellen Anwendungen für ähnliche Aufgabenstellungen wie bei der sogenannten Maus.

Intern arbeiten Joysticks mit zwei kreuzweise angeordneten Potentiometern, deren Widerstandswert sich also durch Bewegen des Steuerknüppels in X- bzw. Y-Richtung stetig ändern läßt. Meist ist auch ein zusätzlicher Druckknopf als Tastenschalter vorhanden, um irgend etwas auszulösen, sobald man mit dem Knüppel ein Objekt auf dem Bildschirm bzw. den Cursor an die gewünschte Stelle gebracht hat.

Es hängt von der Joystick-Interface-Auslegung im Computer ab, welchen Widerstands-Endwert die beiden Potentiometer besitzen müssen, meist aber 50...150 kΩ. Um ohne teuren A/D-Wandler auszukommen, verwendet man gern zwei Monoflops, bei denen die beiden Potentiometer zeitbestimmend wirken. Wenn der Computer die Stellung des Knüppels abfragen will, sendet er über eine I/O-Leitung einen Impuls, um beide Monoflops zu triggern (auszulösen) und stellt mittels einer Zählschleife im Programm die Verzögerungszeiten der beiden Monoflops fest. Da es ICs gibt, die zwei (NE 556) oder sogar vier Timer (NE 558) in einem Gehäuse enthalten, ist auch für den Anschluß von zwei Joysticks (also vier Potentiometern) nur ein relativ geringer Hardware-Aufwand notwendig.

2.3.3 Grafik-Tablets und Lichtgriffel

Die schon beschriebene „Maus" eignet sich zwar recht gut zum Bewegen des Cursors auf dem Bildschirm, für die Eingabe feiner Grafik-Strukturen ist sie aber weniger geeignet. Dafür gibt es sogenannte Grafik-Tablets.

Im wesentlichen haben sich für Grafik-Tablets zwei Arbeitsprinzipien durchgesetzt, die hier kurz beschrieben werden sollen.

Am preiswertesten herstellbar ist ein Grafik-Tablet durch Verwendung eines flächenhaft verteilten Widerstands. Der Anschluß an den Computer erfolgt genauso wie bei einem Joystick über zwei Analog-Eingänge für X- und Y-Koordinaten. *Abb. 2.3.3.1* zeigt die praktische Ausführung eines solchen Geräts. Der Kontakt zwischen X- und Y-Widerstandsbahnen wird geschlossen, sobald man mit einem Stift auf die Oberfläche drückt. Zusammen mit der nötigen Software ist eine sehr komfortable, menügesteuerte Eingabe von Grafiken möglich *(Abb. 2.3.3.2)*.

Da es nicht leicht ist, die nötige Linearität der für solche Tablets nötigen Widerstände über eine größere Fläche zu gewährleisten, verwendet man für Geräte, bei denen es auf höhere Präzision ankommt, ein anderes Prinzip. Das Grafik-Tablet besteht hierbei aus Drahtschleifen, die in X- und Y-Richtung angeordnet sind. In diese Schleifen werden nacheinander kurze elektrische Impulse geleitet, die ein lokales Magnetfeld erzeugen. Dieses impulsartige Feld erzeugt in dem Stift, der mit dem nötigen Interface mit einer Zuleitung verbunden ist, wieder einen kurzen elektrischen Impuls, dessen zeitliche Lage die Position des Stifts auf der Unterlage ergibt.

Denkt man sich die impulsartigen magnetischen Felder eines solchen Tablets ersetzt durch einen Bildschirm, der von einem Elektronenstrahl abgetastet wird, und ersetzt man ferner die Aufnahmespule im Stift durch eine Fotodiode, so gelangt man zum Prinzip des Lichtgriffels: Indem man ihn auf der Bildschirm-Oberfläche

Abb. 2.3.3.1 Ein einfaches Grafik-Tablett (Koala-Pad)

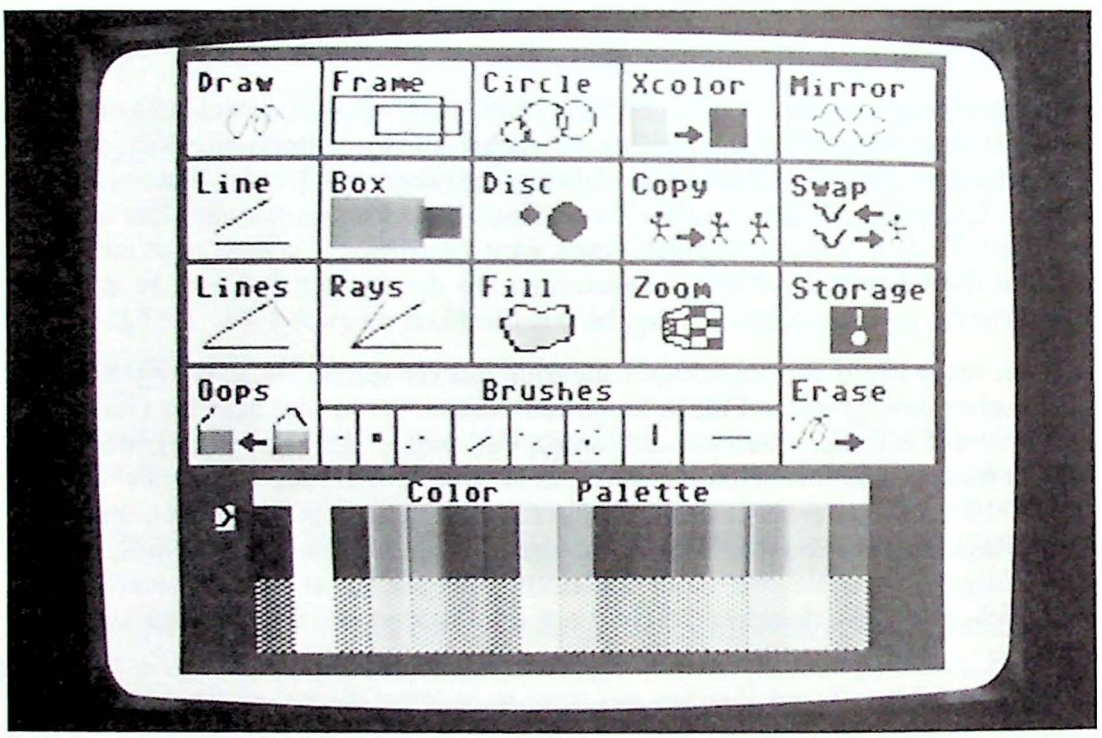


Abb. 2.3.3.2 Die zum Koala-Pad gehörige Software unterstützt das Erstellen von Grafiken auf dem Bildschirm

hin- und herbewegt, kann man ebenfalls Grafiken erzeugen. Die Fotodiode im Stift liefert immer dann einen kurzen Impuls, wenn der Elektronenstrahl gerade den darunterliegenden Punkt der Bildröhre beschreibt.

Der Lichtgriffel setzt voraus, daß erstens die Bildröhren-Leuchtschicht eine extrem kurze Nachleuchtdauer aufweist (Fernseh-Bildröhren sind geeignet, viele Daten-Monitore jedoch nicht) und zweitens die zu erstellende Grafik dunkel auf hellem Hintergrund geschrieben wird, da die Fotodiode nur vor hellem Hintergrund Impulse liefern kann. Das Lichtgriffel-Interface besteht letztlich aus zwei Zählern für die Zeilen und die Punkte innerhalb einer Zeile; der Zählerstand beim Erkennen eines Impulses von der Fotodiode entspricht unmittelbar den X- und Y-Koordinaten des Lichtgriffels.

2.3.4 Spracherkennung

Im Gegensatz zur Spracherzeugung bzw. -speicherung ist die Spracherkennung bzw. Sprachanalyse noch wenig genutzt. (Spracherkennung bedeutet hier, Silben oder Buchstaben in eine äquivalente Zeichenfolge umzusetzen. Demgegenüber versteht man unter Sprecher-Erkennung das Zerlegen des Sprachspektrums in für eine bestimmte Person typische Charakteristika.)

Folgende Schwierigkeiten sind bei der Sprachanalyse bisher noch nicht gemeistert:

1. Die Erkennung von Worten funktioniert nur bei einer bestimmten Person, die den Computer vorher „trainiert" hat.
2. Die zu erkennenden Worte dürfen nicht wie gewohnt zusammenhängend, sondern müssen mit deutlichen Pausen gesprochen werden, da sie sonst vom Rechner nicht isoliert werden können.
3. Die Spracherkennung ist nur mit einem begrenzten, vorher trainierten Wortschatz möglich.
4. Ähnlich klingende Worte werden oft schlecht unterschieden, da dem Rechner meist die Fähigkeit fehlt, den Sinn aus dem Zusammenhang zu erkennen.
5. Je nach Umfang des trainierten Wortschatzes und Geschwindigkeit der Verarbeitung kann die Erkennung eines Wortes etwa eine Sekunde dauern.

Es ist klar, daß bei diesen Einschränkungen die Sprachanalyse nur bei einigen speziellen Anwendungen überhaupt in Frage kommt. Insbesondere die Sprecher-Abhängigkeit ist ein bisher nicht befriedigend gelöstes Problem.

Nulldurchgangs-Analyse

Prinzipiell sind drei Verfahren zur Sprachanalyse üblich. Am einfachsten läßt sich die sogenannte „Nulldurchgangs-Zählung" realisieren *(Abb. 2.3.4.1)*. Dabei wird

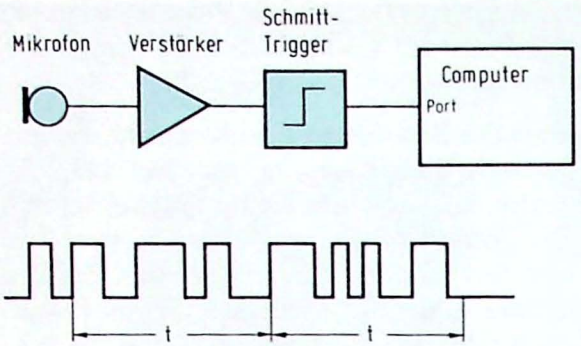


Abb. 2.3.4.1
Spracherkennung durch Zählung von Nulldurchgängen pro Zeiteinheit

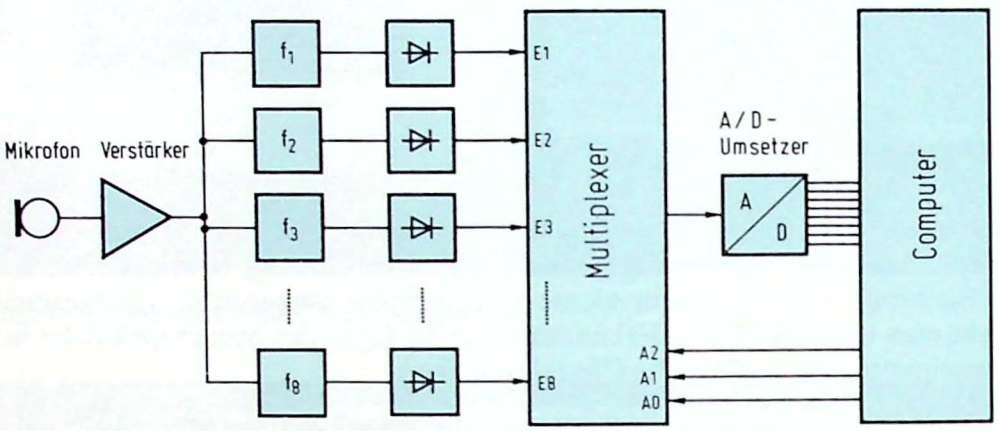


Abb. 2.3.4.2 Spracherkennung durch Zerlegen des Spektrums in mehrere Teilbereiche

das Sprachsignal verstärkt und zu einem Rechtecksignal begrenzt. In regelmäßigen Zeitabständen (t) wird nun die mittlere Frequenz gemessen; dies kann der angeschlossene Computer leicht über einen I/O-Port mit Hilfe eines kurzen Maschinen-Programms tun. Die erste Messung wird gestartet, sobald die Mikrofon-Ausgangsspannung ausreicht, den Schmitt-Trigger anzusteuern (Wortbeginn). Nun kann der Computer für jedes trainierte Wort eine Tabelle anlegen, wie die Frequenzverteilung über die Zeit aussieht. Ein Wort gilt dann als erkannt, wenn das trainierte Muster mit dem des gerade gesprochenen Wortes etwa übereinstimmt.

Hauptnachteil dieser Methode ist, daß ähnlich klingende Worte sehr schlecht unterschieden werden können, weil durch die Begrenzung der Amplitude auf Rechteckform ein wesentlicher Teil der Information vor der eigentlichen Analyse verlorengeht.

Frequenzband-Aufspaltung

Ein zweites Verfahren verwendet z.B. acht Filter, die das Sprach-Spektrum in acht Teile aufspalten (z.B. 300...500 Hz, 500...750 Hz, 750...1000 Hz, 1...1,5 kHz, 1,5...2 kHz, 2...2,5 kHz, 2,5...3 kHz, 3...3,5 kHz).

Jedem dieser acht Kanäle ist ein eigener Gleichrichter zugeordnet. Über einen Multiplexer (z.B. CMOS-Analogschalter) werden die acht Gleichrichter-Ausgänge zyklisch mit einem A/D-Umsetzer verbunden, so daß der Computer nacheinander die Amplituden aller Kanäle abfragen kann *(Abb. 2.3.4.2)*.

Das Programm im Computer speichert nun während der Trainings-Phase für jedes Wort eine Amplituden-Tabelle ab, die z.B. aus 64 Byte besteht: Je ein Byte enthält einen Amplitudenwert der acht Kanäle, so daß acht Bytes in acht Zeitabschnitten gespeichert werden müssen. Bei der späteren Spracherkennung werden wiederum die gespeicherten Tabellenwerte mit dem Muster des gesprochenen Wortes verglichen.

Analyse mit Signalprozessor

Prinzipiell ist es natürlich auch möglich, das verstärkte Mikrofonsignal über einen A/D-Umsetzer direkt zu einem I/O-Port des Computers zu leiten und die Analyse des Frequenzspektrums dem Computer selbst durchführen zu lassen (FFT = Fast-Fourier-Transformation). Damit das mit der nötigen Geschwindigkeit möglich ist, wurden spezielle Signal-Prozessoren entwickelt, deren Befehls- und Registersätze speziell an den Anforderungen der FFT-Analyse orientiert sind (z.B. TMS 320). Herkömmliche Mikroprozessoren sind für diese Aufgabe zu langsam und würden zu unzumutbaren Rechenzeiten führen *(Abb. 2.3.4.3)*.

Die Sprachanalyse mit Signalprozessoren hat sicher die größten Zukunftschancen, da sich die nötigen Bauelemente gut integrieren lassen und mit geeigneten Programmen eine sehr zuverlässige Erkennung möglich ist.

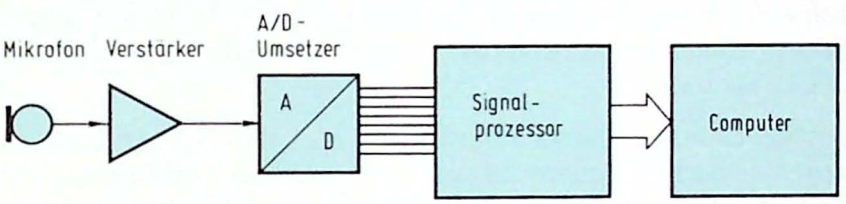


Abb. 2.3.4.3 Mit einem Signalprozessor kann eine Spracherkennung durch direkte Analyse der Schwingungsformen realisiert werden

Daß Sie sich dieses Buch zugelegt haben, beweist, daß Sie sich nicht nur oberflächlich mit Mikrocomputertechnik befassen. Nun kann ein Buch natürlich Technik zwar grundlegend und ausführlich, nicht aber tagesaktuell bringen.

Nun, für alles gibt es das optimale Medium. Die richtige Ergänzung zum „Arbeitsbuch Mikrocomputer" ist die monatlich erscheinende Mikrocomputer-Zeitschrift mc. mc bringt jeden Monat neu aktuelle Informationen über Markttrends, neue Produkte und eben über alles, worüber man in der Branche gerade spricht.

Computer ohne Software sind wertlos. Für Sie als Computerbesitzer bringt mc jeden Monat zahlreiche System- und Anwender-Programme in unterschiedlichen Programmiersprachen wie Assembler, Basic, Pascal und Forth. Natürlich gut kommentiert und ausgiebig getestet. Software nimmt mit Abstand am meisten Seiten in jedem mc-Heft ein.

Falls Sie an vorderster Front der Technik selbst den Lötkolben schwingen: Unter Fachleuten ist die in mc oft sogar mit Platinen-Layouts vorgestellte Hardware berühmt. Der legendäre Einplatinen-Computer EMUF, der mc-CP/M-Computer, der mc-68000-Rechner und andere Meilensteine sind nur ein paar Beispiele. Und was besonders wichtig ist: Diese Hardware wird von mc langfristig unterstützt. mc-Rechner sind keine Eintagsfliegen.

Außerdem finden Sie in mc Grundlagen-Beiträge, in denen elementare Prinzipien von Hard- und Software verständlich erläutert werden — vom Prozessor-Modell bis zum Multitasking-Betriebssystem. Eine Zeitschrift kann schneller auf Markttrends reagieren als ein Buch — deshalb kommen auch Sie um mc nicht herum.

Zugegeben, mc ist nicht jedermanns Sache. Niveau und Tiefe der Beiträge erfordern ernsthaftes Interesse. mc ist keine „Computer-Illustrierte" mit seitenfüllenden bunten Bildern, sondern bringt kompakte, direkt anwendbare Information — Zeile für Zeile, Seite für Seite.

An einigen Stellen in diesem Buch werden Sie über den Namen „mc" stolpern — bei der ersten deutschen Mailbox, bei ersten Versuchen, Strichcode auch für den Abdruck von Programmen einzusetzen, bei Einplatinen-Computern, aber auch am Schluß bei den Literaturhinweisen. Das ist kein Zufall. Seit dem ersten Heft im Mai 1981 konnte mc dank einer kompetenten Redaktion regelmäßig mit innovativen Ideen aufwarten, die später oft von anderen aufgegriffen und kopiert wurden. Aber eben erst später.

Verbinden Sie die Ausführlichkeit des Mikrocomputer-Arbeitsbuches mit der Aktualität einer Zeitschrift für Mikrocomputer-Profis — zwei Medien, die sich sinnvoll ergänzen. mc erscheint jeden Monat neu im gleichen Verlag wie dieses Buch und ist auf dem gleichen fachlichen Niveau geschrieben. Ein kostenloses Probeexemplar liegt für Sie bereit: Franzis-Verlag, Postfach 37 01 20, 8000 München 37. mc erhalten Sie aber auch jeden Monatsanfang bei Ihrem Fachzeitschriften-Händler oder bei der nächsten Bahnhofs-Buchhandlung.

3 Programmierung von Mikrocomputern

3.1 Maschinensprache

3.1.1 Das Monitorprogramm

Über die einzelnen Befehle, die je nach CPU-Typ zur Verfügung stehen, war schon im Abschnitt über Mikroprozessoren die Rede. Die CPU versteht bekanntlich nur binäre Zahlen als Befehle, die der einfacheren Lesbarkeit halber meist in hexadezimaler Form geschrieben werden. Diese Befehle müssen irgendwo im Speicher stehen. Wie sie der Programmierer bei einem von ihm selbst entwickelten Programm dahin bringt, soll nun geklärt werden.

Beim Einschalten des Mikrocomputers ist der RAM-Inhalt (Arbeitsspeicher) zunächst undefiniert. Was die CPU jetzt zu tun hat, muß daher im Festwertspeicher (ROM) stehen, dessen Inhalt beim Aus- und Einschalten nicht verloren geht.

Nun kann man dafür sorgen, daß das Programm im ROM folgende Aufgaben erfüllt:

1. Abfrage einer Tastatur und Decodieren von Kommandos.
2. Ansteuerung eines Video-Interface oder einer Anzeige, auf der hexadezimale Ziffern darstellbar sind.
3. Ausführen von Benutzer-Kommandos:
   a) Anzeige von Speicherinhalten,
   b) Eingabe neuer Werte in Speicherzellen,
   c) Sprung zu einer bestimmten RAM-Adresse (Start eines Anwender-Programms).

Ein Programm, das diese Aufgaben erfüllt, nennt man Monitorprogramm. „Monitor" hat hier nichts mit einem Bildschirm zu tun, sondern kommt von dem englischen Wort „to monitor" (überwachen): Das Programm dient u.a. zum Prüfen von Speicher-Inhalten. Die Ein- und Ausgabe von Adressen und Daten erfolgt gewöhnlich hexadezimal. Dies mag zunächst für einen dezimal denkenden Normalmenschen ungewohnt sein; man sollte sich aber nicht unnötigerweise anstrengen, jede Hexzahl in einen Dezimalwert umzurechnen, um sie sich besser vorstellen zu können, da letzterer im allgemeinen gar nicht interessiert.

Nicht alle Computer besitzen solch ein Monitorprogramm im ROM. Beispielsweise enthalten Rechner, die von vornherein mit einem Floppy-Disk-Laufwerk

ausgestattet sind, nur ein „Urladeprogramm" zum Laden eines Betriebssystems von Disk (dazu später). Andere, insbesondere Heimcomputer, sind primär z.B. für die Programmiersprache Basic ausgelegt, und ein Monitorprogramm, das die Eingabe von Maschinenprogrammen erlaubt, muß bei Bedarf von Kassette oder Floppy geladen werden.

Als Beispiel für ein Monitorprogramm sei das des Apple-II erwähnt. Es meldet sich auf dem Bildschirm mit einem Stern am linken Zeilenrand („Prompt"), um anzuzeigen, daß es bereit ist zur Entgegennahme von Kommandos. Alle Ein- und Ausgaben erfolgen hexadezimal, jede Eingabe muß mit der Return-Taste abgeschlossen werden.

E000.E01F	Speicherinhalt der Adressen E000 bis E01F ausgeben
300:A2 00 60	Ab Adresse 0300 die drei Bytes A2, 00, CA eingeben
300G	Programm an der Adresse 0300 starten
300<320.330V	Speicherinhalb ab 0300 vergleichen mit 320 bis 330, nicht übereinstimmende Adressen werden angezeigt
300<320.330M	Der Speicherbereich 0320 bis 0330 wird in den Bereich 0300 bis 0310 kopiert. (Dieses Kommando funktioniert nur dann korrekt, wenn sich beide Bereiche nicht überschneiden!)
300.37FW	Speicherbereich 0300 bis 037F auf Kassette speichern
800.87FR	Bereich 0800 bis 087F von Kassette lesen
300L	Ab Adresse 0300 zwanzig Zeilen disassembliert ausgeben
CTRL-E	CPU-Registerinhalte ausgeben
I	Bildschirmausgabe auf Inversdarstellung umschalten
N	Bildschirmausgabe auf Normaldarstellung zurückschalten
CTRL-Y	Sprung zur Adresse 03F8. Wenn dort ein Sprungbefehl zum Beginn des Anwenderprogramms steht, läßt sich dieses damit einfach starten
3 CTRL-P	Bildschirm-Ausgabe auf die Karte in Slot 3 umleiten
1 CTRL-K	Tastatureingabe auf die Karte in Slot 1 umschalten

Darüber hinaus erfüllt ein im ROM gespeichertes Monitorprogramm eine weitere wichtige Aufgabe. Es stellt nämlich häufig benötigte Programmstücke als Unterprogramme zur Verfügung, so daß diese vom Anwender in seinen Programmen nicht immer wieder neu programmiert werden müssen, sondern durch einen Unterprogramm-Sprungbefehl aufgerufen werden können. Die wichtigsten dieser Routinen sind die für die Ein- und Ausgabe von Zeichen; sie sind für einige Computertypen hier aufgeführt:

Computer-typ	Zeichen-ausgabe	Zeichen-eingabe	Hex-Byte-ausgabe
Apple-II	FDED	FD0C	FDDA
AIM-65	E9BC	E993	EA46
C-64	FFD2	FFCF	—
TRS-80/I	0033	002B	—
CP/M(60K)	EA0C	EA09	—

Man beachte, daß von den aufgeführten Rechnern nur Apple-II und AIM-65 ein Monitorprogramm im eigentlichen Sinn besitzen. Deshalb gibt es bei den anderen auch keine Routine zur hexadezimalen Ausgabe. Die Zeichenübergabe erfolgt stets im A-Register (Akku). Bei CP/M-Rechnern ist im Normalbetrieb das Urlader-EPROM abgeschaltet, sobald das Betriebssystem geladen ist; die Adressen EA09 und EA0C liegen deshalb im Gegensatz zu den anderen genannten Rechnern im RAM-Bereich.

3.1.2 Hexadezimale Programmeingabe

Mit einem einfachen Monitorprogramm muß man Anwenderprogramme in Form hexadezimaler Befehle und Daten eingeben. Die dafür je nach CPU-Typ zur Verfügung stehenden Befehle wurden bereits im Kapitel über Mikroprozessoren besprochen.

Bei der Entwicklung eines Programms in Maschinensprache geht man zweckmäßig folgendermaßen vor:

1. Das Problem muß in eine zeitliche Abfolge zerlegt werden, z.B. mit Hilfe eines Flußdiagramms oder Struktogramms (siehe später).
2. Nun kann man die nötigen Befehle mnemonisch (LDA, STA usw.) auf ein Blatt Papier schreiben und von Hand unter Zuhilfenahme einer Tabelle in hexadezimale Codes übersetzen.
3. Nach Eingabe der Codes in den Rechner wird man zunächst mit größter Wahrscheinlichkeit feststellen, daß das Programm noch nicht funktioniert, und nötigenfalls die erforderlichen Korrekturen anbringen (das nennt man „Debugging").

Darüber hinaus muß man natürlich wissen, in welchem Adressenbereich überhaupt Anwenderprogramme eingegeben werden dürfen. Einerseits muß in diesem Bereich RAM vorhanden sein, andererseits darf er nicht für systeminterne Zwecke schon verwendet sein (z.B. Video-RAM oder vom Monitorprogramm verwendete Speicherzellen). Man muß also die Speicheraufteilung seines Computers kennen. Für einige Computertypen wird sie daher hier aufgeführt.

Apple-II

Beim Apple-II *(Abb. 3.1.2.1)* sind entweder 48 oder 64 KByte RAM vorhanden. Die 64-KByte-Version unterscheidet sich von der 48-KByte-Version nur dadurch, daß „hinter" dem ROM-Bereich (D000...FFFF) auch RAM vorhanden ist. Durch Ansprechen bestimmter Adressen (C08X) kann wahlweise das ROM oder RAM in diesem Bereich aktiviert und dadurch z.B. eine alternative Programmiersprache geladen werden. Beim Einschalten ist stets zunächst das ROM aktiviert. Im Bereich 0000...BFFF ist RAM vorhanden.

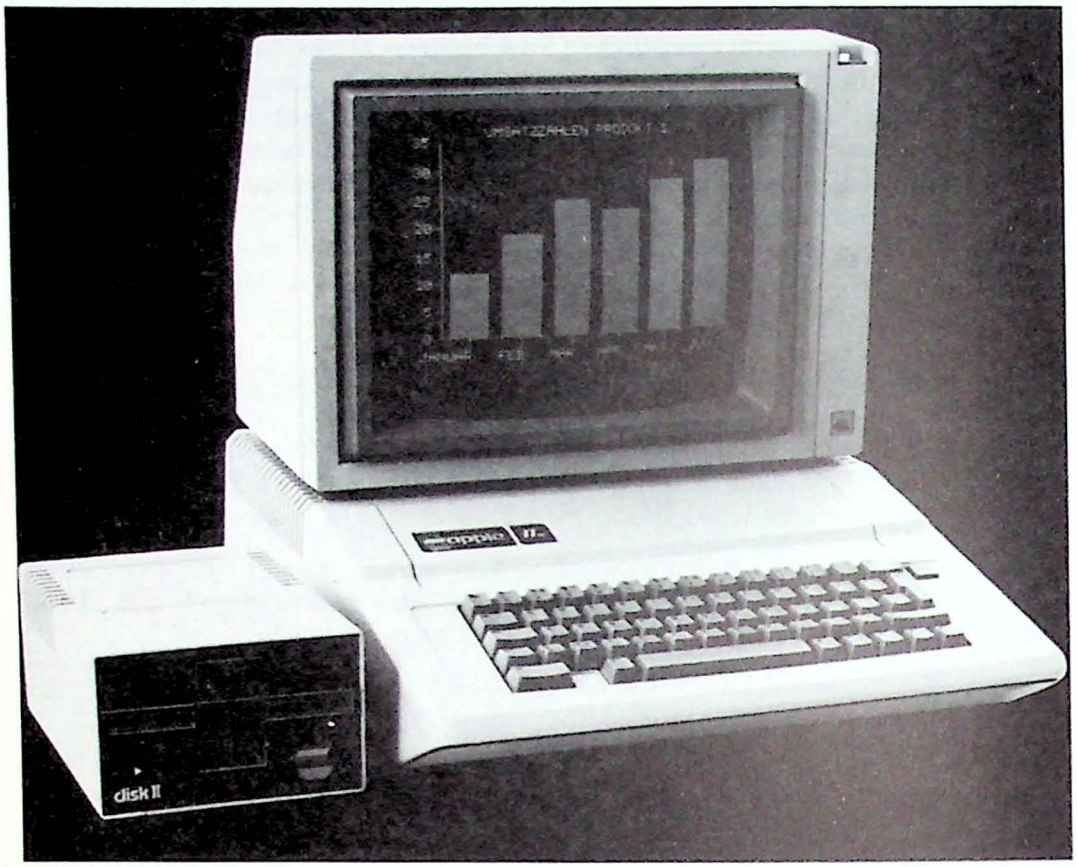


Abb. 3.1.2.1 Der Apple-IIe mit Schwarzweiß-Monitor und 5,25-Zoll-Floppy-Laufwerk

Der RAM-Bereich 0000...00FF wird als Zero Page bezeichnet und von Monitor- und Basic-Interpreter (hierzu später mehr) teilweise genutzt:

0020 ...0049 004E...0055	Vom Monitorprogramm verwendet
0000 ...0005 000A...0018 0050 ...00F8	Vom Applesoft-Basic-Interpreter verwendet

Auch der übrige RAM-Bereich steht dem Anwender nur teilweise zur Verfügung. Der Stack-Bereich 0100...01FF kann natürlich gleichermaßen vom Monitorprogramm wie von Anwenderprogrammen genutzt werden.

0200 ...02FF	Zeilenpuffer für Tastatureingabe
0300 ...03CF	frei
03D0...03EF	von DOS 3.3 verwendet (Disk-Betriebssystem)
03F0, 03F1	BRK-Vektor

03F2, 03F3	Warmstart-Vektor (nach Reset)
03F4	Prüf-Byte (Warm/Kaltstart-Unterscheidung)
03F5 ... 03F7	JMP-Befehl, wird beim Basic-Befehl „&" ausgeführt
03F8 ... 03FA	JMP-Befehl, wird beim Monitor-Befehl CTRL-Y ausgeführt
03FB ... 03FD	JMP-Befehl für NMI-Routine
03FE, 03FF	IRQ-Vektor
0400 ... 07FF	Video-RAM für Textdarstellung
0800 ... 1FFF	frei
2000 ... 3FFF	Video-RAM für Grafik-Seite 1
4000 ... 5FFF	Video-RAM für Grafik-Seite 2
6000 ... 95FF	frei
9600 ... BFFF	vom Disketten-Betriebssystem DOS 3.3 verwendet

Der Bereich C000...CFFF dient als I/O-Bereich. Der Apple-II besitzt acht Slots, die von 0 bis 7 gezählt werden und mit Erweiterungskarten bestückt werden können. Auf der Grundplatine (Motherboard) sind für jeden Slot 256 Bytes Adressenraum für PROMs bzw. EPROMs decodiert, außer für Slot 0, der für die Erweiterung von 48 KByte auf 64 KByte mit einer 16-KByte-RAM-Karte reserviert ist. Ferner stehen für jeden Slot (auch für Slot 0) 16 Bytes im Bereich C080... C0FF zur Verfügung, die speziell für I/O-Aufgaben vorgesehen sind:

Slot 0: C080 ...C08F
Slot 1: C090 ...C09F, C100...C1FF
Slot 2: C0A0...C0AF, C200...C2FF
usw. bis
Slot 7: C0F0 ...C0FF, C700...C7FF

Der Adressenbereich C000...C07F ist für interne I/O-Aufgaben reserviert. Die Tastatur-Daten können bei C000 abgefragt werden; wenn hierbei Bit 7 = 1 ist (höchstwertiges Bit), dann wurde eine Taste gedrückt. Durch Ansprechen der Adresse C010 (z.B. mit LDA oder BIT) wird das Bit-7-Flipflop zurückgesetzt. Dem eingebauten Lautsprecher kann durch Ansprechen der Adresse C030 ein einzelner „Knack" entlockt werden; geschieht das oft und schnell genug hintereinander, wird ein Ton erzeugt.

Im Bereich D000...EFFF folgt schließlich der Basic-Interpreter und in F000... FFFF das Monitorprogramm. Es wurde im letzten Kapitel schon erwähnt, daß sich eine Reihe von Routinen daraus nützlich auch in Anwenderprogrammen einsetzen lassen, indem man sie mit dem Unterprogramm-Aufruf-Befehl JSR anspringt:

F941	Bytes in A- und X-Register als vier Hex-Ziffern ausgeben
FC58	Text-Bildschirm löschen
FCA8	Zeitverzögerungs-Routine. Die Zeit ist vom Akku-Inhalt A abhängig: t = (A*A*5+27*A+26)/2 µs
FDED	ASCII-Zeichen im Akku ausgeben. Dabei erfolgt ein Sprung zur Ausgaberoutine zu jener Adresse, die in den Zellen 0036 und 0037 als Ausgabe-Vektor steht

FDF0	ASCII-Zeichen im Akku unter Umgehung des genannten Vektors auf den Bildschirm ausgeben
FDDA	Byte im Akku als zwei Hex-Ziffern über den Ausgabe-Vektor ausgeben
FDE3	Niederwertige 4 Bits im Akku als Hex-Ziffer ausgeben
FD0C	Sprung über den Eingabe-Vektor 0038/0039 zur Eingabe-Routine: ASCII-Zeichen einlesen und im Akku abliefern
FD1B	Zeichen unter Umgehung des Eingabe-Vektors von der Tastatur holen
FD6A	ASCII-Zeichen in 0033 als Prompt anzeigen und Zeile bis Return in den Puffer ab 0200 einlesen
FD67	Wie FD6A, vorher wird jedoch ein Return-Zeichen (CR) ausgegeben, um den Cursor in die nächste Zeile zu bringen
FD6F	Wie FD6A, jedoch ohne Prompt-Ausgabe
FF2D	„ERR" und Bell-Zeichen (Piepton, Code 07) ausgeben

Die Zeichen-Ein- und Ausgabe erfolgt beim Apple-II im Gegensatz zu den meisten anderen Rechnern mit Bit 7 = 1 (also mit gesetztem höchstwertigem Bit). Ist Bit 7 = 0, so wird das Zeichen invers auf dem Bildschirm dargestellt.

Wenn das Disketten-Betriebssystem DOS 3.3 geladen wird, lenkt es die Ein- und Ausgabevektoren um. Dadurch ist es möglich, DOS-Kommandos von einem Maschinenprogramm auszugeben, indem man der entsprechenden Zeichenfolge ein CTRL-D-Zeichen (Hexcode 84) voranstellt und sie über die Routine FDED ausgibt.

Wie beim Apple-II konkret eine kurze Routine mit Hilfe des Monitor-Programms eingegeben werden kann, soll im folgenden an einem Beispiel gezeigt werden. Die Aufgabe lautet schlicht, vierzig Ausrufezeichen in einer Zeile auf den Bildschirm auszugeben.

Lösen wir zunächst das Problem, nur ein einziges Ausrufezeichen auszugeben. Die Ausgaberoutine an der Adresse FDED des Monitorprogramm-ROMs gibt das im Akku der CPU stehende ASCII-Zeichen aus. Ein Ausrufezeichen besitzt, wie sich anhand der ASCII-Tabelle weiter vorn feststellen läßt, den Code 21. Die Befehlsfolge für die Ausgabe eines Ausrufezeichens lautet also, mnemonisch geschrieben:

```
LDA #$21    ;Wert 21 in den Akku. Ein Doppelkreuz bedeutet „Adressierungsart Immediate", d.h. hinter dem LDA folgt nicht eine Adresse, von der der Wert geholt werden soll, sondern der Wert selbst. Das Dollar-Zeichen bedeutet, daß dieser Wert hexadezimal angegeben ist.
JSR $FDED   ;Unterprogramm-Aufruf der Routine an der hexadezimalen Adresse FDED zur Ausgabe des ASCII-Zeichens im Akku.
RTS         ;Mit diesem Befehl wird diese Routine beendet, es erfolgt ein Rücksprung zum Monitorprogramm.
```

Die nur zum Verständnis geschriebenen Kommentare stehen hier hinter Strichpunkten, damit Sie immer wissen, was noch zum Befehl gehört und was nur eine

Anmerkung ist. Diese Konvention werden Sie ebenso wie die Dollar- und Doppelkreuz-Zeichen im Kapitel über Assembler wiederfinden.

Dieses Mini-Programm ist tatsächlich lauffähig. Wenn wir die Operationscode-Tabelle zur CPU 6502 zu Hilfe nehmen, so steht bei LDA unter der Adressierungsart „Immediate" der Befehlscode A9 und bei JSR der Code 20. Nun müssen wir uns noch erinnern, daß bei 6502-Befehlen, die Adressen enthalten, stets zuerst das nieder- und dann erst das höherwertige Byte der Adresse geschrieben wird. Also:

```
LDA #$21      ergibt A9 21
JSR $FDED     ergibt 20 ED FD
RTS           ergibt 60
```

Mit dem Apple-Monitorprogramm kann man diese Befehlsfolge an irgendeiner freien Stelle im Arbeitsspeicher eingeben, zum Beispiel ab Adresse 0300. (Nach dem Einschalten des Apple-II kommt man mit CALL- 151 vom Basic-Interpreter in den Monitor, so daß ein Stern als Prompt erscheint.) Die Eingabe des Programms erfolgt durch Eintippen der Adresse, eines Doppelpunkts, der sechs obigen Bytes (jeweils untereinander durch einen Leerraum getrennt) und schließlich Drücken der Return-Taste:

```
0300:A9 21 20 ED FD 60 (Return)
0300G (Return)
```

Der Befehl 0300 G startet das Programm. Jetzt muß auf Ihrem Bildschirm wirklich ein Ausrufezeichen erscheinen und dann sofort wieder das Monitor-Prompt-Sternchen. Allerdings: Das Ausrufezeichen erscheint invers, weil im Apple für Normaldarstellung Bit 7 = 1 gesetzt werden muß! Die nötige Korrektur ist einfach: Man braucht nur A1 statt 21 einzugeben ($21 + $80 = $A1). Nachträglich läßt sich das falsche Byte so ändern:

```
0301:A1 (Return)
```

Der Rücksprung mit RTS zum Monitorprogramm funktioniert übrigens, weil beim Apple der Befehl G wie ein Unterprogramm-Aufruf wirkt, d.h. aus dem Monitorprogramm erfolgt ein JSR zu der vor dem G geschriebenen Adresse (hier 0300). Insgesamt haben wir also schon zwei Unterprogramm-Ebenen verwendet: einmal diejenige des G-Kommandos und dann die des Befehls JSR $FDED.

Eigentlich wollten wir aber nicht ein, sondern vierzig Ausrufezeichen auf den Bildschirm bringen. Man könnte nun einfach die gerade geschriebene Befehlsfolge LDA #$21, JSR $FDED vierzigmal nacheinander in den Speicher schreiben und schließlich mit RTS beenden. Das würde aber immerhin 201 Bytes „fressen". Besser ist es, die fünf Bytes mit einer Programmschleife zu umrahmen, die vierzigmal durchlaufen wird. Dafür benötigt man irgendein Register als Zähler, zum Beispiel das X-Register der CPU 6502 (mit Y funktioniert es genauso, wichtig ist nur, daß die Ausgabe-Routine bei FDED den Inhalt von X bzw. Y nicht verändert, was hier jedoch gewährleistet ist).

Es ist prinzipiell egal, ob man das Zählregister zuerst mit Null lädt und dann bis vierzig aufwärts zählt oder ob man es mit 40 lädt und bis Null abwärts zählt. Im letzteren Fall spart man sich allerdings einen Vergleichsbefehl, ob 40 erreicht ist, deshalb zählen wir lieber abwärts:

```
        LDX #$28        ;X-Register mit dezimal 40 = hex 28 laden
HIERHER LDA #$A1        ;Code für Ausrufezeichen in den Akku
        JSR $FDED       ;ASCII-Zeichen im Akku ausgeben
        DEX             ;X-Register herunterzählen (dekrementieren)
        BNE HIERHER     ;Sprung zur Adresse HIERHER, falls nicht Null
        RTS             ;andernfalls zurück zum Monitorprogramm
```

Wenn von der Adresse HIERHER die Rede ist, dann ist das natürlich keine hexadezimale Adresse, sondern nur ein vorläufig als Adressen-Ersatz hingeschriebenes Symbol (Label). Denn welcher Befehl an welcher Adresse steht, weiß man ja erst später, wenn man die mnemonischen Befehle in hexadezimale Codes umgewandelt hat, und die 6502-Befehle können ein, zwei oder drei Bytes lang sein.

Das Nachschlagen in der 6502-Operationscode-Tabelle ergibt folgende Hex-Codes und, da damit auch die Befehlslänge bekannt ist, Adressen für das Programm:

Label	Mnemonisch	Adr.	Hex-Code
	LDX #$28	0300	A2 28
HIERHER	LDA #$A1	0302	A9 A1
	JSR $FDED	0304	20 ED FD
	DEX	0307	CA
	BNE HIERHER	0308	D0 ??
	RTS	030A	60

Die zwei Fragezeichen stehen für die bei Branch-Befehlen wie BNE nötige relative Sprungadresse. Das bedeutet, hinter BNE kann leider nicht die absolute Adresse von HIERHER geschrieben werden (0302), sondern die Differenz zwischen der auf den Branch-Befehl folgenden Adresse 030A und dem Sprungziel 0302. Zusätzlich ist zu beachten, daß negative Relativ-Adressen (also solche, bei denen wie hier rückwärts gesprungen werden soll) im Zweierkomplement dargestellt werden, d.h. alle Bits werden invertiert und 1 addiert. Das klingt kompliziert, ist aber in der Praxis einfach.

Man braucht nämlich nur, beginnend beim Byte 60, das bei 030A steht, hexadezimal rückwärts die Bytes bis zum Sprungziel abzuzählen: 00, FF, FE, FD, FC, FB, FA, F9, F8 — und schon ist HIERHER erreicht. Das richtige Byte hinter D0 muß also F8 heißen. Insgesamt sieht die einzutippende Bytefolge dann so aus:

```
0300:A2 28 A9 A1 20 ED FD CA D0 F8 60
```

Nach Eingabe von 0300 G erscheinen tatsächlich vierzig Ausrufezeichen auf dem Bildschirm! Ein kleiner Vorgriff auf das Kapitel über Disassembler: Wenn Sie

jetzt beim Apple 300 L eingeben, dann können Sie die Hex-Codes in ihrer mnemonischen Form bewundern.

AIM-65

Ein zweites Beispiel für einen Computer auf der Basis der CPU 6502, der ebenfalls ein recht komfortables Monitorprogramm im ROM besitzt, ist der AIM-65 von Rockwell. Er besitzt in der Grundversion nicht ein Video-Interface, sondern ein 20stelliges alphanumerisches LED-Display, einen ebenso breiten Thermodrucker und eine ASCII-Tastatur. An RAM sind auf der Platine 4 KByte untergebracht.

Mit dem AIM-65 kompatibel sind die beiden Rechner PC-100 (Siemens bzw. System-Elektronik) und MC-65 (64 KByte dynamisches RAM, Terminal-Schnittstelle). Sie weisen sogar die gleiche Adressenbelegung wie das Vorbild auf und brauchen daher nicht getrennt behandelt zu werden.

Folgende Monitorprogramm-Kommandos stehen zur Verfügung:

E	Initialisierung des eingebauten Texteditors
T	Eintritt in den Editor, ohne den Text zu löschen
N	Falls Assembler-PROM installiert, Assembler starten
5	Falls Basic-PROMs installiert, Basic-Kaltstart
6	Falls Basic-PROMs installiert, Basic-Warmstart
*	Adresse eingeben
P, A, X, Y, S	Prozessorregister ändern (P = Status, S = Stackpointer usw.)
R	Prozessor-Registerinhalte anzeigen
M	Speicherinhalte ab angegebener Adresse anzeigen
Space	Die nächsten vier Bytes anzeigen (Space = Leertaste)
/	Speicherinhalte überschreiben
I	Mnemonische Befehlseingabe
K	Speicherinhalt disassembliert anzeigen
G	Programm an der mit „*“ spezifizierten Adresse starten
Z	Einzelschritt-Modus mit Disassembler-Anzeige ein-/ausschalten
V	Einzelschritt-Modus mit Register-Anzeige ein-/ausschalten
H	Programmzähler-Protokollierung für Einzelschritt-Modus
?	Breakpoint-Adressen anzeigen
#	Alle Breakpoints löschen
B	Breakpoints setzen/löschen
4	Breakpoint-Modus ein-/ausschalten
L	Speicherinhalt z. B. von Kassette laden (IN = T, F = Name, T = 1)
D	Speicherinhalt z. B. auf Kassette speichern (OUT = T, F = Name, T = 1)

CTRL-PRINT	Thermodrucker ein-/ausschalten
PRINT	Anzeigeninhalt drucken
LF	Drucker-Zeilenvorschub
1	Motor von Kassettenrecorder 1 ein-/ausschalten
2	Motor von Kassettenrecorder 2 ein-/ausschalten
3	Prüfen einer Kassettenrecorder-Aufzeichnung auf Lesbarkeit
F1	Sprung zur Adresse 010C
F2	Sprung zur Adresse 010F
F3	Sprung zur Adresse 0102

Da der AIM-65 vor allem zur Entwicklung von Maschinenprogrammen konstruiert wurde, besitzt er ein sehr komfortables Monitorprogramm. Neu ist vielleicht der Begriff „Breakpoint". Wenn an einer bestimmten Adresse ein Breakpoint gesetzt wird, so wird der ursprünglich dort stehende Operationscode durch einen BRK-Befehl (hex 00 beim 6502) ausgetauscht. Sobald das Anwenderprogramm dorthin gelangt, springt der Prozessor über den IRQ-Vektor ins Monitorprogramm, das dann z.B. den Inhalt aller Register an dieser Stelle ausgeben kann. Das kann bei der Fehlersuche sehr nützlich sein.

Eine Besonderheit ist auch der mit dem Kommando I aufrufbare Line-by-Line-Assembler (Zeilenassembler). Er gestattet es, Programme in mnemonischer Form einzugeben, also z.B. BNE F8 statt D0 F8. Allerdings kann man im Gegensatz zu einem „richtigen" Assembler Adressen nicht symbolisch schreiben, z.B. wie beim Apple-II beschrieben BNE HIERHER.

Ebenso wie beim Apple-II steht beim AIM-65 nur ein Teil des RAM-Bereichs beim Anwender frei zur Verfügung. In der Zero-Page sind folgende Adressen vom System verwendet:

00DF...00AC	Vom Monitorprogramm verwendet
0004 ...00AC	Vom optionalen Assembler verwendet
0000 ...00DB	Vom optionalen Basic-Interpreter verwendet

Im Stack-Bereich (0100...01FF) sind einige Speicherzellen ebenfalls für Systemzwecke reserviert, z.B. Sprungvektoren für die Funktionstasten F1...F3 und Ein-Ausgabe-Vektoren. In der 4-KByte-RAM-Version steht dann ab 0200 Arbeitsspeicher bis zur Adresse 0FFF zur Verfügung. Der Raum von hex 1000 bis 9FFF ist für externe Speicher-Erweiterungen vorgesehen.

Im Bereich AXXX sitzen die Peripherie-Bausteine des AIM-65, u.a. zwei VIA-Bausteine 6522 und ein RIOT 6532:

A000...A00F	VIA 1 (frei für Anwender)
A400...A47F	RAM im RIOT 6532 (vom System verwendet)
A480...A49F	I/O (vom System verwendet) und Timer im RIOT
A800...A80F	VIA 2 (vom System verwendet)

Der optionale 8-KByte-Basic-Interpreter sitzt im Bereich B000...CFFF, der ebenfalls nachrüstbare 4-KByte-Assembler in D000...DFFF. Das immerhin 8 KByte umfassende Monitorprogramm enthält u.a. auch einen Texteditor zur Eingabe z.B. von Assembler-Programmen sowie den erwähnten Line-by-Line-Assembler; es residiert im Bereich E000...FFFF.

Die wichtigsten Monitor-Unterprogramme beim AIM-65 besitzen folgende Adressen:

E182	Warmstart des Monitorprogramms
E394	„ERROR“ ausgeben
E3FD	Hex-Byte als zwei ASCII-Zeichen einlesen
E6E1	Drucker ein- und ausschalten
E7A3	„FROM=“ ausgeben und Adresse in A41C, A41D einlesen
E7A7	„TO=“ ausgeben und Adresse in A41C, A41D einlesen
E7DC	Zuletzt angezeigtes Zeichen auf dem Display löschen
E848	Den Benutzer nach dem gewünschten Eingabegerät fragen
E871	Den Benutzer nach dem gewünschten Ausgabegerät fragen
E8FE	Eingabe auf die Tastatur zurückschalten
E93C	Ein ASCII-Zeichen von der Tastatur holen
E95F	Ein ASCII-Zeichen von der Tastatur holen und anzeigen, DEL erlaubt
E973	Ein ASCII-Zeichen von der Tastatur holen und anzeigen
E9F0	Return und Line Feed ausgeben (CRLF)
EA13	Display löschen
EBDB	Ein Zeichen von der seriellen Schnittstelle holen
EEA8	Ein Zeichen an die serielle Schnittstelle ausgeben
EEFC	Ein Zeichen auf Display, ggf. auf Drucker ausgeben
EF05	Ein Zeichen auf Display ausgeben
F000	Ein Zeichen auf den Thermodrucker ausgeben
F6CF	Texteditor-Warmstart

Die Baudrate der seriellen TTY-Schnittstelle wird durch den Inhalt der Speicherzellen A417 und AA18 bestimmt:

Bit/s	A417	A418
300	0C	C0
1200	02	FD

Nun zurück zu dem schon beim Apple-II verwendeten Beispiel, vierzig Zeichen auszugeben, beim AIM-65 sinnvollerweise auf den Thermodrucker, da das einzeilige Display nur 20 Zeichen darstellen kann. Statt der Apple-Ausgabe-Routine FDED muß für den AIM-65 also F000 eingesetzt werden. Ferner braucht das höchstwertige Bit des auszugebenden Zeichens hier nicht gesetzt zu werden, als Code für das Ausrufezeichen wird also hex 21 statt A1 gewählt. In mnemonischer Form, zunächst noch mit dem Label HIERHER als Schleifen-Rücksprungziel, sieht die AIM-Version daher so aus:

Label	Mnemonisch	Adresse	Hex-Code
	LDX #40	0300	A2 28
HIERHER	LDA #$21	0302	A9 21
	JSR $F000	0304	20 00 F0
	DEX	0307	CA
	BNE HIERHER	0308	D0 F8
	JMP $E182	030A	4C 82 E1

Man beachte, daß das Programm hier nicht mit einem RTS-Befehl endet, sondern mit einem JMP-Befehl zum Monitor-Warmstart. Dies ist nötig, weil das zum Starten des Programms normalerweise verwendete Kommando G nicht mit JSR zum Anwenderprogramm springt, sondern mit JMP, dieses also nicht als Unterprogramm betrachtet. Wie beim Apple-II kann das X-Register als Zähler verwendet werden, da sein Inhalt durch die Druckroutine F000 nicht verändert wird. Auch die Adresse 0300, ab der das Programm beim Apple eingegeben wurde, kann für den AIM-65 beibehalten werden, da auch er an dieser Stelle freies RAM besitzt.

Die Eingabe des Programms kann z.B. mit dem Zeilenassembler des AIM-65 erfolgen, wodurch man sich das lästige Nachsehen der Operationscodes in einer Tabelle erspart. Dieser Zeilenassembler ist auch in der Lage, die nach dem Branch-Befehl nötige Relativadresse F8 automatisch zu errechnen, wenn man nach BNE die absolute Adresse des Sprungziels (hier 0302) schreibt:

```
<*>=300                 ;Anfangsadresse wählen
<I>                     ;Zeilenassembler aufrufen
0300  LDX #28           ;Werte müssen hexadezimal eingegeben werden,
0302  LDA #21           ;ohne Dollar-Symbol!
0304  JSR F000          ;Die Kommentare in dieser Spalte werden
0307  DEX               ;natürlich nicht eingegeben, ebenso
0308  BNE 0302          ;die Adressen ganz links, sie werden
030A  JMP E182          ;vom Rechner selbst erzeugt.
```

Wenn man nun die Escape-Taste drückt, ist man wieder im normalen Monitor-Modus. Der Programmstart erfolgt dann so:

```
<*>=300
<G>/
```

Jetzt druckt der Thermodrucker (sofern er eingeschaltet ist) vierzig Ausrufezeichen aus, allerdings natürlich in zwei Zeilen, weil er ja nur zwanzig Zeichen breit ist.

Die korrekte Eingabe des Programms läßt sich auch hier mit einem Disassembler kontrollieren:

```
<K>*=300        ;Anfangsadresse
/06             ;Zu disassemblierende Zeilenzahl (dezimal!)
```

Dann erscheint das Programm in jenem mnemonischen Format, wie es mit dem Kommando I vorher eingegeben wurde. Übrigens hatten die früheren Apple-Versionen ebenfalls einen Line-by-Line-Assembler, dieser fiel jedoch später dem Ladeprogramm für Disketten zum Opfer (Autostart-Monitor).

C-64

Der von Commodore gebaute C-64 arbeitet mit der CPU 6510, die eine Spezialversion des 6502 darstellt: Sie besitzt den gleichen Befehlssatz, aber einige zusätzliche I/O-Leitungen, die beim C-64 u.a. dafür verwendet werden, die Adressenkonfiguration umzuschalten, damit die eingebauten 64 KByte RAM überhaupt genutzt werden können. Die dafür nötigen I/O-Register der CPU 6510 sind in der Zero Page untergebracht:

0000	Datenrichtungs-Register für Prozessor-Port
0001	Datenregister für Prozessor-Port

Außer dem kleinen Bereich 00FB...00FF werden alle Zero-Page-Zellen vom Basic-Interpreter verwendet. Ein Monitorprogramm ist im C-64-ROM nicht eingebaut. Die Entwickler haben das Gerät nämlich in erster Linie als Heimcomputer und nicht als Entwicklungssystem für Maschinenprogramme konzipiert und gaben sich deshalb auch keine Mühe, einen größeren Zero-Page-Bereich für Anwenderprogramme freizuhalten.

Der übrige Adressenbereich des C-64 wird recht verschachtelt von I/O-Bausteinen, RAM und ROM belegt. RAM ist in der beim Einschalten vorhandenen Normal-Konfiguration von 0800 bis 9FFF und von C000 bis CFFF frei verfügbar:

0100 ...01FF	Stack
0200 ...0258	Zeilenpuffer für Eingabe
0259 ...0276	Für IEC-Gerätedaten reserviert
0277 ...0280	Tastaturpuffer (wird von Interrupt-Routine gefüllt)
0281 , 0282	Vektor für Basic-RAM-Beginn
0283 , 0284	Vektor für Basic-RAM-Ende
0285 ...0287	Timeout-Flag, Farbwerte
0288	Höherw. Teil der Video-RAM-Anfangsadresse
0289	Länge des Tastaturpuffers
028A	Flag für Tastatur-Repeatfunktion
028B ...0301	für verschiedene Zwecke verwendet
0302 , 0303	Vektor für Basic-Warmstart
0304 ...030F	für verschiedene Zwecke verwendet
0312 , 0313	Vektor für USR-Befehl in Basic
0314 , 0315	IRQ-Vektor
0316 , 0317	BRK-Vektor
0318 , 0319	NMI-Vektor
031A ...03FE	für verschiedene Zwecke verwendet
0400 ...07FF	Video-RAM für Textdarstellung

0800 ...9FFF	Arbeitsspeicher für Basic-Programme und Daten
A000 ...BFFF	8 KByte ROM mit Basic-Interpreter
C000 ...CFFF	4 KByte RAM z. B. für Maschinenprogramme
D000 ...D7FF	Video-Controller (VIC) und Tongenerator (SID)
D800 ...DBFF	RAM für Farbwerte (parallel zum Text-Video-RAM)
DC00...DFFF	I/O-Bereich (CIA)
E000 ...FFFF	8 KByte ROM mit I/O-Routinen

Damit sich bei ROM-Änderungen durch den Hersteller die Adressen der wichtigsten Unterprogramme im ROM für Anwenderprogramme nicht ändern, liegt eine „Sprungtabelle" am oberen ROM-Ende. Sie besteht aus JMP-Befehlen, die zu den eigentlichen Startadressen dieser Unterprogramme führen. Die JMP-Befehle sitzen an folgenden Adressen:

FFB1	Video-Controller (VIC) initialisieren
FF84	CIA-Bausteine initialisieren
FF87	RAM testen
FF8A	I/O initialisieren
FF8D	Vektoren für Ein- und Ausgabe initialisieren
FF90	Status initialisieren
FF93	Sekundäradresse nach LISTEN senden
FF96	Sekundäradresse nach TALK senden
FF99	RAM-Ende setzen
FF9C	RAM-Beginn setzen
FF9F	Tastatur abfragen
FFA2	Timeout-Flag für IEC-Bus setzen
FFA5	Eingabe vom IEC-Bus
FFA8	Ausgabe auf IEC-Bus
FFAB	UNTALK senden
FFAE	UNLISTEN senden
FFB1	LISTEN senden
FFB4	TALK senden
FFB7	Status holen
FFBA	Dateiparameter setzen
FFBD	Parameter für Dateinamen setzen
FFC0	OPEN
FFC3	CLOSE
FFC6	Eingabegerät setzen
FFC9	Ausgabegerät setzen
FFCC	IEC-Bus schließen
FFCF	ASCII-Zeichen holen (in den Akku)
FFD2	ASCII-Zeichen im Akku ausgeben
FFD5	LOAD-Befehl
FFD8	SAVE-Befehl
FFDB	Uhr setzen
FFDE	Uhrzeit holen

FFE1 Stoptaste abfragen
FFE4 Tastatur abfragen (ohne Warten)
FFE7 Kaltstart durchführen
FFEA Uhrzeit weiterzählen
FFED Zahl der Zeilen und Spalten für Bildschirm setzen
FFF0 Cursor setzen bzw. Position holen
FFF3 CIA-Startadresse in X und Y laden

Die Routinen für LOAD, SAVE, OPEN, CLOSE, LISTEN, TALK, UNLISTEN und UNTALK beziehen sich dabei auf Ein- und Ausgabe mit dem speziellen Commodore-Bus, der unter Beibehaltung des IEC-Bus-Protokolls nicht mit paralleler, sondern serieller Übertragung arbeitet. Er ist dadurch zwar langsamer, benötigt aber weniger Leitungen.

Zurück zu unserem einfachen Beispiel, vierzig Ausrufezeichen auf den Bildschirm auszugeben. Sinnvollerweise schreibt man das Maschinenprogramm ab Adresse C000 in den C-64-Speicher, da es dort nicht versehentlich von einem Basic-Programm überschrieben werden kann (der Bereich C000...CFFF ist für den Basic-Interpreter tabu). Die Routine zur Ausgabe eines Zeichens hat die Start-Adresse FFD2. Das nötige Programm für den C-64 sieht also so aus:

Label	Mnemonisch	Adresse	Hex-Code
	LDX #40	C000	A2 28
HIERHER	LDA #$21	C002	A9 21
	JSR $FFD2	C004	20 D2 FF
	DEX	C007	CA
	BNE HIERHER	C008	D0 F8
	RTS	C00A	60

Das Programm kann hier mit RTS beendet werden, weil der SYS-Befehl in Basic, mit dem es gestartet werden kann, wie bei JSR die Rückkehradresse auf den Stack schreibt.

Wie bekommt man in den C-64 nun aber ein Maschinenprogramm hinein? Es ist ja kein Monitorprogramm im ROM vorhanden. Glücklicherweise ist es aber nicht schwierig, den Basic-Interpreter dazu zu bringen, hexadezimale Werte an eine beliebige Stelle im Speicher zu bringen. *Abb. 3.1.2.2* nimmt ein kleines Basic-Programm vorweg, das diese Aufgabe übernimmt.

Der Einfachheit halber sind hier als DATA-Anweisungen schon die Hex-Codes des gerade entwickelten Mini-Programms eingetragen. Natürlich wäre auch ein Programm vorstellbar, das die Operationscodes nicht hexadezimal, sondern dezimal enthält; dann müßte man aber erst jeden einzelnen Hex-Wert von Hand in sein dezimales Äquivalent umrechnen. Diese Aufgabe übernimmt das kleine Programm gleich mit.

Die Methode, Maschinenprogramme mit einem Basic-Ladeprogramm zu generieren, ist auch bei Computern sinnvoll, die sehr wohl ein Monitorprogramm besit-

```
70REM *** HEX-PROGRAMM LADEN ***
80AA=49152:REM ANFANGSADR.HEX C000
90A=AA:REM ZAEHLER = ANFANGSADR.
100READH$:IFH$="-"THENEND
110Z$=LEFT$(H$,1):GOSUB200:D=16*Z
120Z$=MID$(H$,2):GOSUB200:D=D+Z
130POKEA,D:A=A+1:GOTO100
200IFZ$>"9"THENZ=ASC(Z$)-55:RETURN
210Z=VAL(Z$):RETURN
220REM HEX-BYTES, ENDE = "-"
230DATA A2,28,A9,21,20,D2,FF
240DATA CA,D0,F8,60,-
```

Abb. 3.1.2.2
Basic-Hilfsprogramm zum Laden eines Speicherbereichs mit hexadezimalen Bytefolgen

zen — nämlich dann, wenn für eine bestimmte Anwendung sowohl ein Basic- als auch ein davon aufgerufenes Maschinenprogramm nötig ist. Dann braucht man nicht zwei Programme getrennt von Kassette oder Floppy-Disk zu laden, sondern nur das Basic-Programm, das den Maschinencode in Form von DATA-Anweisungen gleich enthält und nach dem Start irgendwo im Speicher ablegt. Während beim C-64 ab C000 ein Speicherbereich zur Verfügung steht, der für Basic tabu ist, muß bei anderen Rechnern (z.B. beim Apple-II oder AIM-65) erst dafür gesorgt werden, daß das Basic-Programm und seine Variablen nicht das erzeugte Maschinenprogramm überschreiben können. Das geschieht durch Festlegen der oberen Basic-RAM-Grenze mit Hilfe des dafür vorgesehenen Vektors; darüber mehr im Kapitel über Basic.

Der Start des erzeugten Maschinenprogramms kann beim C-64 mit dem Befehl SYS erfolgen (beim Apple-II lautet der äquivalente Befehl CALL). Ihm muß die Startadresse folgen, allerdings dezimal. Was ist nun C000 dezimal? Bei der Umrechnung kann der Basic-Interpreter behilflich sein, wenn man sich erinnert, daß im Hexadezimal-System die Wertigkeit einer Stelle 16mal diejenige der rechts davon stehenden ist. Da die Hex-Ziffer C der Dezimalzahl 12 entspricht (hex 0...F entspricht dezimal 0...15), kann man den SYS-Befehl auch so schreiben:

SYS 12*16*16*16 + 0*16*16 + 0*16 + 0*1

oder noch einfacher, da nur die höchstwertige Hex-Ziffer ungleich Null ist:

SYS 12*16*16*16 oder
SYS 49152

Alternativ wäre die Umrechnung auch mit einem kleinen Basic-Programm möglich *(Abb. 3.1.2.3)*.

Wenn Sie einen der genannten SYS-Befehle eingeben, erscheinen auf dem Bildschirm vierzig Ausrufezeichen — es funktioniert also wirklich. Allerdings wäre der Versuch fehlgeschlagen, wenn wir statt hex 21 für ein Ausrufezeichen den Code hex 61 für ein kleines a gewählt hätten: Der C-64 arbeitet intern nämlich leider nicht mit der Standard-ASCII-Codetabelle. Der Zeichensatz läßt sich durch gleich-

```
1REM *** Z$ = FELD MIT HEX-ZIFFERN ***
10DIMZ$(16):FORI=0TO15:READZ$(I):NEXT
20DATA0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
22:
23REM *** HEX- ODER DEZ.-ZAHL EING.***
25PRINT"1=DEZ>HEX 2=HEX>DEZ Q=QUIT ";:GETA$
27PRINTA$:IFA$="1"THENINPUT"DEZ:";D:GOSUB35
28IFA$="2"THENINPUT"HEX:";H$:GOSUB75
29IFA$<>"Q"THEN25
30END
35H$="":REM *** D IN H$ UMRECHNEN ***
40D=D/16:H$=Z$(16*(D-INT(D)))+H$
50D=INT(D):IFD>0THEN40
60PRINT"HEX:";H$:RETURN
70:
75REM *** H$ IN D UMRECHNEN ***
80D=0:P=1:FORI=LEN(H$)TO1STEP-1
90FORN=0TO15:IFZ$(N)=MID$(H$,I,1)THEN94
92NEXT:PRINT"NICHT HEX!":RETURN
94D=D+P*N:P=P*16:NEXTI
96PRINT"DEZ:";D:PRINT:RETURN
```

Abb. 3.1.2.3
Basic-Programm zum Umrechnen dezimaler in hexadezimale Zahlen und umgekehrt

zeitigen Druck auf Shift und die Commodore-Taste (links unten) zwischen zwei Belegungen umschalten:

Codebereich (hex)	Einschalt-zustand	Nach Umschaltung
00...1F	Steuerzeichen	Steuerzeichen
20...3F	ASC 20...3F	ASC 20...3F
40...5F	ASC 40...5F	ASC 60...7F
60...7F	Grafikzeichen	ASC 40 ...5F

Mit ASC sind die äquivalenten ASCII-Codebereiche gekennzeichnet. Man erkennt, daß beim Einschalten zunächst keine Kleinschrift möglich ist und nach dem Umschalten auf gemischte Groß- und Kleinschrift die Codes für Groß- und Kleinbuchstaben vertauscht sind. Im Steuerzeichen-Bereich entspricht praktisch nur Return (hex 0D) dem ASCII-Standard. Für die Kommunikation mit anderen Rechnern (z.B. über ein Telefonmodem) ist also unbedingt eine Konvertierung der C-64-Codes erforderlich.

3.1.3 Assembler

Das Nachsehen von Operationscodes und die Umrechnung symbolischer Adressen (Labels) in absolute ist besonders bei längeren Programmen umständlich und zeitraubend. Deshalb verwendet man für die Erstellung von Maschinenprogram-

men meist ein Hilfsprogramm, das Assembler heißt und diese Aufgaben übernimmt.

Das Programm selbst wird zunächst mit Labels und mnemonischen Befehlen geschrieben. Dazu benötigt man einen Text-Editor, der die Eingabe dieses sogenannten „Quelltexts" (englisch Source = Quelle) erlaubt.

Der eigentliche Assembler arbeitet meist in zwei Durchläufen (2-Pass-Assembler). Zunächst werden alle symbolischen Adressen in absolute umgerechnet. Dazu legt der Assembler in einem reservierten Speicherbereich eine Symbol-Tabelle an. Sie enthält alle Namen und die zugehörigen Absolutwerte. Stellt der Assembler im ersten Durchlauf z.B. fest, daß ein und derselbe Name versehentlich zweimal im Programm als symbolische Adresse auftaucht, gibt er eine entsprechende Fehlermeldung aus, z.B. „Duplicate Label". Auch die Korrektheit der Schreibweise mnemonischer Anweisungen wird in Pass 1 kontrolliert. (Diese Aufgabe kann allerdings auch schon der Texteditor übernehmen, wenn er speziell für den Assembler geschrieben wurde.)

Im zweiten Durchlauf (Pass 2) generiert der Assembler dann das Maschinenprogramm (den sogenannten Objektcode), wobei er für Sprungbefehle u.a. absolute Adressen aus der Symboltabelle ermittelt. Dabei können u.U. weitere Fehler im Quelltext festgestellt und gemeldet werden, z.B. eine inkorrekte Schreibweise eines Labels hinter Sprungbefehlen.

Je nach Assembler und Speicherplatz im jeweiligen Computer gibt es unterschiedliche Möglichkeiten, woher (Quelltext) und wohin (Objektcode) assembliert wird. Lediglich die Symboltabelle steht normalerweise immer im RAM. Der Sourcecode kann zeilen- oder blockweise aus dem RAM oder von Kassette bzw. Floppy-Disk assembliert werden. Ebenso kann der Objektcode, also das erzeugte Maschinenprogramm, entweder sofort ins RAM oder zunächst auf Floppy oder Kassette generiert werden. Letztere Methode ist insbesondere bei sehr langen Programmen sinnvoll, wenn Sourcecode, Objektcode und Symboltabelle nicht gleichzeitig im Arbeitsspeicher Platz finden. Zudem gibt es bei vielen Assemblern die Möglichkeit, das erzeugte Maschinenprogramm im RAM zunächst nicht an der dafür eigentlich vorgesehenen Adresse abzulegen, wo es auch lauffähig ist und gestartet werden kann, sondern an einer anderen Adresse. Dies ist nötig, wenn das erzeugte Programm just in dem Adressenbereich laufen soll, den der Assembler selbst mit seinem Code belegt oder in dem der Quelltext oder die Symboltabelle liegt.

In aller Regel sind Assembler selbst in Maschinensprache geschrieben, um eine möglichst hohe Geschwindigkeit zu erzielen. 100 Quelltext-Zeilen werden dann in weit weniger als einer Sekunde assembliert. Im folgenden werden fünf unterschiedliche Assembler vorgestellt — als kleine Auswahl der auf dem Software-Markt verfügbaren.

Lisa

Der für den Apple-II von Lazer Systems entwickelte 6502-Assembler „Lisa" (Lazer Systems Interactive Symbolic Assembler) besitzt einen eigenen Texteditor, der schon bei der Quelltext-Eingabe elementare Syntax-Prüfungen durchführt (Syntax = Schreibweise). Lisa residiert im Adressenbereich hex 7000...9A00; vor dem Start muß beim Apple-DOS 3.3 der Befehl MAXFILES1 eingegeben werden, damit der Assembler nicht teilweise von DOS-Datenpuffern überschrieben wird. Der Quelltext wird ab hex 1800 bis max. 5FFF abgelegt. Dabei werden alle Mnemonics wie LDA, JSR, DEX usw. in einem einzigen Byte codiert gespeichert (Tokens), alles andere wird in ASCII abgelegt. Die Symboltabelle wird von 6000...7FFF erzeugt. Für den Objektcode sind die Bereiche 0300...03CF und 0800...17FF reserviert. Die Kaltstart-Adresse ist 7000 (der Quelltext wird dabei gelöscht), die Warmstart-Adresse 7003. Nach dem Start stehen folgende Kommandos zur Verfügung:

I	Zeilen eingeben bzw. an Quelltext anfügen, Ende = CTRL-E + Return
I 5	Neue Zeile(n) vor Zeile 5 anfügen, Ende mit CTRL-E und Return
D 5	Zeile 5 löschen
D 5,7	Zeilen 5 bis 7 löschen
L	Quelltext auflisten; Unterbr. = Leertaste, Abbruch = CTRL-C
M 4	Zeile 4 neu eingeben (M 4,8: Zeilen 4 bis 8)
LO N	Quelltext-Datei mit Name = N von Floppy laden (z.B. LO PGM)
SAV N	Quelltext-Datei mit Name = N codiert auf Floppy abspeichern
AP N	Quelltext-Datei von Floppy an den Text im Speicher anhängen
W N	Quelltext-Datei als ASCII-Textfile auf Floppy speichern, Name = N
N	Quelltext löschen (New)
AS	Assemblierung starten (Abkürzung A möglich)
BR	Sprung zum Apple-Monitorprogramm

Wenn man DOS-Befehle eingeben will (z.B. CATALOG, um das Disk-Inhaltsverzeichnis anzusehen), muß man vorher CTRL-D drücken, also z.B. CTRL-D CATALOG (und Return). Dieses Verfahren ist auch anzuwenden, um die Listing-Ausgabe auf den Drucker zu leiten. Wenn die entsprechende Interface-Karte in Slot 1 steckt, so gibt man bei Lisa CTRL-D PR#1 ein. Mit CTRL-D PR#0 wird der Drucker wieder ausgeschaltet. Nützlich ist die Möglichkeit, DOS-Kommandos so einzugeben, auch dann, wenn man eine mit W erzeugte ASCII-Textdatei von Diskette laden möchte; das funktioniert mit CTRL-D EXEC NAME.

Nach dem Kommando I zur Eingabe von Quelltext-Zeilen muß am Beginn jeder Zeile, die nicht mit einem Label beginnt, ein Leerraum stehen. Durch diese Konvention kann der Assembler stets einwandfrei zwischen Labels und Operationscodes unterscheiden, und es ist sogar zulässig, für Symbole Namen zu wählen, die mit 6502-Mnemonics identisch sind (z.B. RTS).

Außer den 6502-Mnemonics stehen folgende sogenannten Pseudo-Befehle zur Verfügung, die den Assembler-Ablauf steuern:

ORG $900	Maschinenprogramm-Startadresse = hex 900
OBJ $1000	Objektcode ab Adresse 1000 ablegen (muß nach ORG stehen)
LB EPZ $F8	Dem Label LB wird die Zero-Page-Adresse F8 zugewiesen
LB EQU $980	Dem Label LB wird die absolute Adresse 0980 zugewiesen
HEX 093042	Ab der aktuellen Adresse werden die Bytes 09 30 42 abgelegt
ASC "NANU"	NANU als ASCII-Zeichenfolge ablegen, Bit 7 = 1
ASC 'NANU'	NANU als ASCII-Zeichenfolge ablegen, Bit 7 = 0
LST	Während des Assemblierens Kontroll-Listing ausgeben
NLS	Kontroll-Listing abschalten
ICL "N"	Nächste Sourcedatei laden, Name = N
ADR LB	Adresse des Labels LB als Low-/High-Byte ablegen
DFS 20	20 Bytes überspringen, d.h. Speicher reservieren
PAU	Assemblierung zu Testzwecken hier unterbrechen
BYT LB	Das Low-Byte der Adresse LB ablegen
HBY LB	Das High-Byte der Adresse LB ablegen
BLT	Branch on Less Than, gleichwertig mit BCC
BGE	Branch on Greater/Equal, gleichwertig mit BCS
END	Ende des Quellcodes

Hinter den eigentlichen Befehlen oder in eigenen Kommentarzeilen kann der Programmierer jeweils hinter einem Strichpunkt Anmerkungen unterbringen, die bei der Assemblierung ignoriert werden.

Argumente hinter Mnemonics (z.B. hinter LDA) können je nach Typ unterschiedlich gekennzeichnet werden. Die Adressierungsart „Immediate" wird außer bei ASCII-Zeichen (' oder ") mit einem Doppelkreuz (#) gekennzeichnet, andere Zahlen oder Labels gelten als Adresse:

$345	Adresse hex 0345
#16	Dezimalzahl 16 (= hex 10)
#$10	Hex-Zahl 10 (= dezimal 16)
%100011	Dualzahl 1000 (= hex 23)
#LB	Low-Byte der Adresse von Label LB
/LB	High-Byte der Adresse von Label LB
'A	ASCII-Zeichen A mit Bit 7 = 0 (hex 41)
"A	ASCII-Zeichen A mit Bit 7 = 1 (hex C1)

Die ursprüngliche Version von Lisa läßt leider die Verwendung einer 80-Zeichen-Karte beim Apple-II nicht zu, sie würde u.a. nach CTRL-C immer abgeschaltet. Dieses Problem läßt sich durch folgende Änderungen umgehen:

Adresse	alt:	neu:
7719	20 93 FE	EA EA EA
771C	20 89 FE	EA EA EA
7C03	20 93 FE	EA EA EA
7C06	20 89 FE	EA EA EA
70AE	59 FF	65 FF
76B7	20 58 FC	20 39 99
9939	XX...XX	A9 0C 20 ED FD 4C 58 FC

In der zuletzt angeführten Änderung sorgt der Code 0C dafür, daß der 80-Zeichen-Bildschirm korrekt gelöscht wird. Ein weiteres Problem bei Lisa ist die unkorrekte Spalten-Zählung bei Ausgabe der Symboltabelle und die um ein Zeichen versetzte END-Zeile; es läßt sich so lösen:

Adresse	alt:	neu:
98A7	20 73 78	EA A2 00
98AA	0D 0D 2A	20 73 78
98AD	2A 20	0D 0D
95D2	0B	0C

Bei der Eingabe des Beispielprogramms aus dem letzten Kapitel in den Lisa-Assembler geht man folgendermaßen vor, wobei natürlich jede Zeile mit Return abzuschließen ist:

	Eingabe	Kommentar
	I	Eingabe beginnen, der Editor erzeugt nun am linken Rand Zeilennummern
1	OUT EQU $FDED ;AUSGABE	Ausgaberoutine als Label definieren
2	ORG $300	Startadresse = hex 0300
4	LDX #40	40 Zeichen ausgeben
4	HIER LDA "! ;BIT 7 = 1	Ausrufezeichen in den Akku
5	JSR OUT ;AUSGEBEN	OUT ist hex FDED (siehe Zeile 1)
6	DEX ;NAECHSTES "!"	
7	BNE HIER	Sprung nach Zeile 4
8	RTS ;ZUM MONITOR	
9	END	Quelltext-Ende
10	Ctrl-E	Verlassen des Editors
	Ctrl-D PR# 1	Ab jetzt Ausgabe auf Drucker
	AS	Quelltext assemblieren

Was jetzt auf den Drucker ausgegeben wird, geht aus *Abb. 3.1.3.1* hervor. Von links nach rechts sieht man die Objektcode-Adressen, den erzeugten Hex-Code, die Quelltext-Zeilennummern, die mnemonischen Befehle sowie ihre Argumente und, sofern vorhanden, Anmerkungen des Programmierers. Am Schluß des Listings folgt die Symboltabelle; sie ist in unserem Beispiel natürlich nicht sehr umfangreich.

```
PASS 1
PASS 2

0800            1  OUT     EQU $FDED              ;AUSGABE
0300            2          ORG $300
0300 A228       3          LDX #40
0302 A9A1       4  HIER     LDA "!                 ;BIT7=1
0304 20EDFD     5          JSR OUT                ;AUSGEBEN
0307 CA         6          DEX                    ;NAECHSTES "!"
0308 D0F8       7          BNE HIER
030A 60         8          RTS                    ;ZUM MONITOR
                9          END

ABSOLUTE VARABLES/LABELS

OUT     FDED   HIER    0302

SYMBOL TABLE  START:6000
SYMBOL TABLE LENGTH:0022
```

Abb. 3.1.3.1
Druckerprotokoll einer Assemblierung mit „Lisa"

Um das erzeugte Maschinenprogramm gleich auszuprobieren, kann man mit dem Kommando BR aus Lisa in das Monitorprogramm gehen und 300 G eingeben. Zum Assembler gelangt man mit 7003 G oder Reset zurück, ohne den Quellcode zu löschen.

Der SC-Assembler

Der von einer in Dallas/Texas beheimateten „Software Company" entwickelte SC-Assembler wurde ebenfalls für den Apple-II entwickelt. Es gibt ihn in zwei Versionen, nämlich für 48-KByte- und für 64 KByte-Geräte. Im letzteren Fall residiert er im RAM, das „hinter" dem Apple-Basic-Interpreter-ROM liegt, und nimmt daher keinen Adressenraum des Arbeitsspeichers in Anspruch:

Version	Assembler	Symboltabelle	Quelltext
SC	1000...31D2	ab 3200 aufw.	ab 9600 abw.
SC.LC	D000...F200	ab 1000 aufw.	ab 9600 abw.

Die Eingabe erfolgt ähnlich wie bei einem Basic-Interpreter: Zeilen, die mit einer Nummer beginnen, werden als Quelltext gespeichert, andere werden als Kommandos interpretiert. Der SC-Assembler beherrscht folgende Kommandos:

ASM	Assemblieren
AUTO 10	Autom. Zeilennr.-Vorgabe ab 10
COPY 2,5,9	Zeilen 2 bis 5 vor Zeile 9 kopieren
DELETE 5,9	Zeilen 5 bis 9 löschen
EDIT 10	Zeile 10 editieren (auch CTRL-E)
FAST	Auf schnelle Assemblierung schalten
FIND ABC	String ABC im Sourcetext suchen
HIDE	Source-Zeilen unsichtbar machen

INCREMENT2	Inkrement 2 für AUTO
LIST	Sourcezeilen listen
LIST 10,30	Sourcezeilen 10 bis 30 listen
LOAD ABC	Quelldatei ABC von Disk laden
MANUAL	AUTO abschalten (Cursor an linken Rand bringen!)
MEMORY	zeigt verwendeten Speicherbereich an
MERGE ABC	Quelldatei ABC von Disk hinzuladen (mischen)
MGO $300	Start eines Maschinenprogramms bei hex 300
MNTR	Sprung zum Monitor
NEW	Sourcetext löschen
RENUMBER	Zeilen neu numerieren
RST $FF65	Reset-Vektor nach Adresse FF65 umlenken
SAVE ABC	Sourcetext mit Name = ABC auf Disk speichern
SLOW	Langsame Assemblierung (zum Mitlesen!)
SYMBOLS	Alphabetisch sortierte Symboltabelle ausgeben
TEXT ABC	Source uncodiert als T-File auf Disk speichern
VAL 32	Umrechnung dezimal/hexadezimal (ergibt hier hex 20)
$	Ausführen eines Monitorprogramm-Befehls, z.B. $300L

DOS-Befehle werden hier im Gegensatz zum Lisa-Assembler ohne vorheriges CTRL-D eingegeben. Normalerweise erfolgt die Abspeicherung von Quelltext auf Disk mit SAVE im gleichen Format wie bei dem früheren Apple-Integer-Basic (Ganzzahl-Interpreter) in codierter Form, mit TEXT dagegen als normale ASCII-Textdatei. Eine solche Datei kann wieder geladen werden, indem man AUTO (und Return) eingibt, mit dem Cursor an den linken Rand geht und EXEC Filename eingibt.

Bei der Eingabe des Quelltextes muß am linken Rand stets eine manuell eingegebene oder mit AUTO erzeugte Zeilennummer stehen. Die Eingabe muß sauber formatiert in Spalten erfolgen, wobei CTRL-I als Tabulator-Taste dient (z.B. um von der Label- in die Mnemonic-Spalte zu gelangen). Zur Ablaufsteuerung steht eine Reihe von Pseudo-Befehlen zur Verfügung:

LB .EQ $345	Dem Label LB wird der Wert hex 0345 zugewiesen
.OR $300	Anfangsadresse des Objektcodes
.TF ABC	Objektcode mit Name = ABC auf Disk generieren, nicht im RAM
.DA	Datenbytes ablegen, z.B. .DA #'X'+$40
.AS 'C'	ASCII-Zeichen C mit Bit 7 = 0 ablegen
.LISTOFF	Assembler-Listing ausschalten
.TI 64,ABC	Listing-Seitenlänge 64 Zeilen, Seitenüberschrift = ABC

Eine Unterscheidung der Zuweisung von Zero-Page- und anderen Labels gibt es hier nicht; der Assembler merkt selbst, wenn eine Adresse in der Zero Page liegt. Zur Kennzeichnung von Immediate, Low- und High-Byte werden wie bei Lisa die Symbole # und / verwendet, ebenso das Dollar-Zeichen für hexadezimale und das Prozentzeichen für duale Zahlen. Kommentare werden jedoch nicht mit

```
:ASM

                0010          .OR $300
FDED-           0020 OUT      .EQ $FDED      AUSGABE
0300- A2 28     0030          LDX #40        40 ZEICHEN
0302- A9 A1     0040 HIER     LDA #'!'+$80   BIT7=1
0304- 20 ED FD  0050          JSR OUT
0307- CA        0060          DEX            NOCHMAL?
0308- D0 F8     0070          BNE HIER       JA!
030A- 60        0080          RTS            ZUM MONITOR


SYMBOL TABLE

0302- HIER
FDED- OUT

0000 ERRORS IN ASSEMBLY
```

Abb. 3.1.3.2
Beispiel für ein mit dem SC-Assembler geschriebenes Programm

einem Strichpunkt gekennzeichnet, sondern mit einem Stern, wenn sie am Zeilenanfang beginnen, oder überhaupt nicht, wenn sie hinter einem Befehl stehen.

Abb. 3.1.3.2 zeigt, wie unser kleines Beispielprogramm als SC-Assembler-Listing aussieht. Ein wenig umständlich ist hier die Erzeugung des Codes für ein Ausrufezeichen mit Bit 7 = 1, da der SC-Assembler dafür keinen speziellen Befehl besitzt. Dafür ist es mit ihm allerdings möglich, fast beliebig lange Programme zu assemblieren, da der Objektcode statt in den Arbeitsspeicher auch auf Diskette geleitet werden kann, wenn man am Quelltext-Beginn den Befehl TF verwendet.

Der Makro-Assembler Big Mac

Auch beim 6502-Assembler Big Mac ist ein passender Texteditor eingeschlossen. Big Mac residiert entweder im RAM von 0803 bis 1D00 oder in der Apple-II-Language-Karte im Adressenbereich ab D000, also „hinter" dem Basic-Interpreter. Er versteht folgende Kommandos:

C	Disk-Inhaltsverzeichnis anzeigen, DOS-Kommandos zulassen
L ABC	Sourcetext mit Name = ABC von Diskette laden (B-File)
R ABC	Textdatei (T-File) namens ABC von Disk laden
S ABC	Sourcetext mit Name = ABC auf Disk speichern (B-File)
W ABC	Sourcetext als ASCII-Datei (T-File) abspeichern, Name = ABC
A ABC	Source-Datei ABC an Sourcetext im Speicher anhängen
D	Auf Laufwerk 2 oder zurück auf 1 schalten
E	Editor (siehe unten)
Z	Wie E, aber alle Tabulator-Positionen werden Null gesetzt
O ABC	Erzeugten Objektcode mit Name = ABC auf Disk speichern
Q	Rücksprung zum Basic-Interpreter

Bei den Disk-Kommandos fragt der Assembler automatisch nach einem Namen, d.h. nach dem Kommando-Buchstaben (z.B. O) ist erst Return zu drücken. Wenn

man mit E in den Editor-Modus gegangen ist, steht eine Reihe von weiteren Kommandos zur Verfügung:

HI	Obergrenze für Sourcetext setzen
NEW	Sourcetext im Speicher löschen
PR#1	Ausgabe auf Slot 1 umleiten
USER	Sprung nach hex 3F5
TABS	Tabulator-Positionen setzen
LEN	Länge des Sourcetextes in Bytes ausgeben
W 5	Position von Zeile 5 im Speicher ausgeben
MON	Sprung zum Monitor (zurück mit CTRL-Y)
TRON	Kommentare am Zeilenende nicht anzeigen
TROFF	TRON wieder aufheben
STRIP	Kommentare nach „*“ oder „;“ löschen
Q	Zur Kommandoebene zurück
SYM	Adressenbereich für Symboltabelle definieren
ASM	Assemblierung beginnen
D 5	Source-Zeile 5 löschen
R 5	Source-Zeile 5 durch neue ersetzen
L	Sourcetext mit Zeilennummern listen
P	Sourcetext ohne Zeilennummern listen
/	Auflistung abbrechen oder fortsetzen
F ABC	Zeichenfolge ABC im Quelltext suchen
C AB,CD	Zeichenfolge AB durch CD ersetzen
COPY 4,9 TO 10	Zeilen 4 bis 9 vor Zeile 10 kopieren
MOVE 4,9 TO 10	Wie COPY, aber dann Zeilen 4 bis 9 löschen
E 4	Zeile 4 editieren (ändern; auch E 4,6 möglich)
A	Neue Zeilen ans Ende des Sourcetextes anhängen
I 6	Neue Zeile(n) vor Zeile 6 einfügen

Sobald man mit A oder I aus dem Editor in den Insert-Modus gegangen ist, kann man (da dies die ursprüngliche Apple-II-Tastatur noch nicht zuließ) mit CTRL-L zwischen Groß- und Kleinbuchstaben umschalten und mit CTRL-X aus einer Eingabezeile in die Editor-Kommandoebene zurückspringen.

Für das Editieren von Quelltext-Zeilen gibt es eine Reihe von nützlichen Steuerzeichen:

CTRL-I	Zeichenfolge bis zu beliebigem CTRL-Zeichen einfügen
CTRL-D	Zeichen unter dem Cursor löschen
CTRL-F X	Cursor zum nächsten Auftreten des Zeichens X bewegen
CTRL-L	Zwischen Groß- und Kleinschrift hin- und herschalten
CTRL-O	Wie CTRL-I, jedoch werden nur CTRL-Zeichen eingefügt
CTRL-R	Ursprüngliche Form der Zeile wiederherstellen
CTRL-Q	Rest der Zeile löschen und Editiervorgang beenden

CTRL-C	Wie CTRL-R, aber Editiervorgang beenden
CTRL-B	Cursor an den Zeilenanfang stellen
CTRL-N	Cursor einen Leerraum nach rechts
Return	Editieren der Zeile beenden

Im Sourcetext erfolgt die Darstellung von Zahlen in folgendem Format dezimal, hexadezimal, dual oder als Label:

#	Adressierungsart Immediate (Zahl, nicht Adresse)
123	Dezimalzahl 123
$43	Hexadezimalzahl 43
%10010	Dualzahl 10010
#<LB,#LB	Niederwertiger Teil der Adresse von Label LB
#>LB,/LB	Höherwertiger Teil der Adresse von Label LB
"Y"	ASCII-Zeichen Y mit Bit 7 = 1
'Y'	ASCII-Zeichen Y mit Bit 7 = 0
LB+$10	Adresse von Label LB plus hex 10
"K"−$40	Code des Zeichens K minus hex 40

Neben den normalen 6502-Mnemonics sind folgende Pseudo-Befehle in Big Mac implementiert:

EQU,=	Zuweisung, z. B. LB EQU $300 oder LB = $300
ORG $300	Anfangsadresse hex 300 setzen (ohne ORG ist sie $8000)
OBJ $380	Objektcode unabhängig von ORG ab $380 in Speicher ablegen
END	Ende des Sourcetextes
LST OFF	Assemblerlisting ausschalten
LST ON	Assemblerlisting einschalten
EXP ON	Makro komplett im Assemblerlisting ausgeben
EXP OFF	Nur Makro-Pseudo-Befehl ausgeben
PAU	Assemblerlisting bis zum nächsten Tastendruck anhalten
PAG	Form-Feed-Zeichen (Blattvorschub) auf Drucker ausgeben
AST 20	20 Sternchen im Listing ausgeben
SKP 5	5 Leerzeilen im Listing einfügen
ASC 'YY'	ASCII-Zeichen YY mit Bit 7 = 0 ablegen
ASC "YY"	ASCII-Zeichen YY mit Bit 7 = 1 ablegen
DA $FDF0	Adresse $FDF0 als Bytefolge F0 FD ablegen
DDB $FDF0	Die zwei Bytes FD F0 (!) im Speicher ablegen
DFB $34,123,21	Die drei Bytes hex 34, dezimal 123 und 21 ablegen
HEX 232105	Die drei Bytes 23 21 05 im Speicher ablegen
DS 20	20 Bytes freihalten bzw. überspringen

Darüber hinaus gibt es noch drei Pseudo-Anweisungen für die sogenannte bedingte Assemblierung:

DO ABC	überspringt den Sourcetext bis ELSE oder (wenn ELSE fehlt) FIN, wenn der Ausdruck ABC Null ist.
ELSE	Der Programmteil von ELSE bis FIN wird nur ausgeführt, wenn der Ausdruck nach DO gleich Null ist.
FIN	beendet den Programmteil bedingter Assemblierung.

Definiert man z.B. gleich am Programmanfang ABC = 0 oder ABC = 1, so kann man dadurch leicht zwei oder mehr unterschiedliche Programmversionen erzeugen (z.B. für mehrere Computertypen mit derselben CPU), ohne umständliche Änderungen im Sourcetext vornehmen zu müssen.

Ferner gibt es die Möglichkeit, sogenannte Makros zu definieren. Eine solche Definition muß mit MAC beginnen und mit EOM enden. Während ein Unterprogramm nur einmal im Speicher steht und bei Bedarf vom sonstigen Programm aus mit JSR angesprungen wird, werden Makros als ganzer Programmteil in den Objektcode eingebaut. Das bedeutet, daß zwar die Makro-Definition nur einmal im Sourcetext steht, aber überall dort, wo sie mit PMC aufgerufen wird, genau so als Objektcode generiert wird.

Der MS-DOS-Assembler MASM-86

Sowohl Intel (ASM-86) als auch Microsoft (MASM) bieten für die Prozessoren 8086 und 8088 geeignete Makro-Assembler an. Das Programmierkonzept unterscheidet sich hierbei deutlich von den bisher aufgeführten Assemblern. Wesentliche Unterschiede ergeben sich nicht nur durch die segmentierte Adressierung der Prozessorfamilie, sondern auch durch das Verständnis von Labeln. Während bei den 6502-Assemblern ein Label immer die damit spezifizierte Adresse beinhaltet, ist dies beim 8086 nur bei Sprungzielen der Fall, die mit einem nachfolgenden Doppelpunkt deklariert werden müssen, z.B. LAB:. Folgt dagegen hinter einem Label (ohne Doppelpunkt) eine Tabelle aus mehreren Bytes, so repräsentiert dieses Label nicht die Anfangsadresse der Tabelle, sondern den Tabellen-Inhalt (!). Da man irgendwie aber auch an die Anfangsadresse herankommen muß, z.B. um das Register SI damit zu laden, gibt es dafür den Operanden-Prefix OFFSET, z.B.:

```
MOV SI,OFFSET TAB2
```

Eine Alternative stellt die Verwendung des Befehls LEA dar (Load effective Address, effektive Adresse laden):

```
LEA SI,TAB2
```

Mit „effektiver Adresse" ist dabei stets die 16-Bit-Adresse bezogen auf den jeweiligen Segment-Beginn gemeint.

Da der Assembler selbst bei manchen Befehlen nicht unterscheiden kann, ob ein Wort (16 Bit) oder ein Byte (8 Bit) gemeint ist, muß man ggf. dem Operanden

```
A>masm exetest
The Microsoft MACRO Assembler , Version 1.25
 Copyright (C) Microsoft Corp 1981,82,83

Object filename (exetest.OBJ):
Source listing  (NUL.LST): exetest
Cross reference (NUL.CRF):

Warning Severe
Errors  Errors
0       0

A>type exetest.lst
                                name Test1
 0000                           stack segment stack ;f.Linker
 0000     14 (                  dw 20 dup(?) ;Platz f.Stack
                ????
                      )

 0028                           stack ends
 0000                           code segment ;Ab hier Pgm.!
                                assume cs:code
 0000  0E                       start: push cs ;Codesegment
 0001  1F                       pop ds ;in DS kopieren
 0002  BA 000D R                mov dx,offset text
 0005  B4 09                    mov ah,9 ;Textausgabe-Funktion
 0007  CD 21                    int 21H
 0009  B4 4C                    mov ah,4CH
 000B  CD 21                    int 21H  ;Programmende
 000D  45 69 6E 20 65 72        text db 'Ein erster Test!$'
       73 74 65 72 20 54
       65 73 74 21 24
 001E                           code ends
                                end start


A>link exetest

   Microsoft Object Linker V2.01 (Large)
(C) Copyright 1982, 1983 by Microsoft Inc.

Run File (EXETEST.EXE):
List File (NUL.MAP):
Libraries (.LIB):


A>exetest
Ein erster Test!
A>
```

Abb. 3.1.3.3
Beispiel für Assemblieren und Linken eines .EXE-Files in MS-DOS

WORD PTR oder BYTE PTR voranstellen, um eine eindeutige Übersetzung zu ermöglichen.

Da der Assembler MASM vorwiegend bei Rechnern mit dem Betriebssystem MS-DOS eingesetzt wird, hier noch einige Bemerkungen zur Programmentwicklung unter MS-DOS.

Das Betriebssystem teilt beim Laden eines Programms diesem normalerweise selbst einen Adressenbereich zu, der mit einem Segmentbeginn anfängt. Dadurch ist das Nennen einer Anfangsadresse im Assembler-Quelltext entbehrlich. Man muß lediglich ein Segment für den Stack, eines für den Programmcode und eventuell eines für Datenbereiche reservieren. Diese „logischen" Segmente müssen nicht notwendigerweise 64 KByte umfassen, sind also nicht als physikalische Segmente zu verstehen.

Nun aber zu einem kurzen Beispiel-Programm in 8086-Assembler *(Abb. 3.1.3.3)*. Es gibt lediglich die Worte „Ein erster Test!" auf dem Bildschirm aus. Der Assembler wird mit MASM und dem Namen der Source-Datei (ohne deren Extension .ASM) aufgerufen und erzeugt das Listing als Datei EXETEST.LST. Das erzeugte Programm EXETEST.OBJ ist aber noch nicht lauffähig; es muß erst mit Hilfe des Linkers (der das Zusammenbinden mehrerer Programme und Segmente gestattet) in ein .EXE-File umgewandelt werden. (EXE-Files sind im Gegensatz zu COM-Files, die immer bei 100H im aktuellen Segment beginnen, frei im Speicher verschiebbar und dürfen auch länger als 64 KByte sein.)

Die Textausgabe ist hier besonders einfach ausgefallen, da die MS-DOS-Funktion zur Ausgabe eines Strings herangezogen wurde. Der Aufruf des erzeugten Programms vom Betriebssystem aus erfolgt wie bei externen MS-DOS-Kommandos einfach durch Eingeben des Namens ohne Extension.

Die folgende Tabelle faßt schließlich die wichtigsten Pseudo-Befehle des 8086-Assemblers zusammen.

Assembler-Pseudobefehle (Intel-ASM-86, Microsoft-MASM)

Anweisung:	Beispiel:	Wirkung:
ASSUME	ASSUME CS:CODESEG	Erzeugen von Segment-Override, falls außerhalb CS
AT	BIOS SEGMENT AT 0	Weist absolute Adresse zu
B	1010B	Kennzeichnet binäre Zahl
BYTE	BUF THIS BYTE	Kennzeichnet Byte (vgl. WORD)
DB	BETA DB ?	Byte-Definition; ? = keine Wertzuweisung
DD	LAB2 DD 'ABCD'	Doppelwort definieren, hier hex 44 43 42 41
DQ	DQ 14329	Definiert vier Worte (define quad word)

Anweisung:	Beispiel:	Wirkung:
DT	DT 'abcdefghij'	Definiert zehn Bytes (define ten bytes)
DUP	DW 20 DUP(?)	Reserviert Speicherplatz, hier 20 undef. Worte
DW	DELTA DW 'AB'	Wortdefinition, hier hex 42 41
END	END START	Ende des Programms
ENDIF	ENDIF	Ende eines IF-Abschnitts
ENDM	ENDM	Ende einer Makro-Definition
ENDP	GETCHR ENDP	Ende einer Prozedur (vor ENDP muß RET stehen!)
ENDS	STACKSEG ENDS	Ende der Deklaration eines logischen Segments
EQU	ZERO EQU 0	Wertzuweisung (auch „=" möglich)
EVEN	EVEN	Setzt IP auf gerade Adresse, ggf. durch NOP-Einfügung
EXTRN	EXTRN CO:NEAR	Linker-Anweisung zur Einbindung externer Prozeduren
FAR	RET FAR	Kennzeichnet Adressierung außerhalb des Segments
GROUP	DGR GROUP DATA	Segment-Gruppen-Definition
H	100H	Kennzeichnet hexadezimale Zahl
INCLUDE	INCLUDE FILE2	Andere Source-Datei einbinden
IF	IF IBM=1	Beginn eines Abschnitts mit bedingter Assemblierung
LABEL	TOS LABEL WORD	Kennzeichnet momentane Adresse TOS als Wort
LENGTH	LENGTH TAB2	Liefert Länge einer Variablen in Bytes bzw. Worten
LOCAL	LOCAL XYZ	Kennzeichnet lokale Variable innerhalb Makro
MACRO	MACRO PRINT	Beginn einer Makro-Definition (Name hier PRINT)
MASK	AND DX,MASK B	Liefert die mit RECORD hier als B definierten Bits
NAME	NAME PGM1	Name des Programm-Moduls an Linker übergeben
NEAR	PR1 PROC NEAR	Kennzeichnet Adressierung innerhalb des Segments
OFFSET	MOV DX,OFFSET TXT	Adressenbezug auf Segmentbeginn
ORG	ORG 32*4	Beginn des erzeugten Codes
PAGE	PAGE 60,120	Seitenformat setzen (hier 60 Zeilen m. je 120 Zeichen)
PROC	GETCHR PROC FAR	Prozedur (mit CALL GETCHR aufzurufen)

Anweisung:	Beispiel:	Wirkung:
PTR	TG EQU WORD PTR (BP)	Kennzeichnet Byte- oder Wort-Zugriff
PUBLIC	PUBLIC DOS	Kennzeichnet ein „public label"
Q	177Q	Kennzeichnet Oktalzahl (auch O statt Q möglich)
PAGE	DATA SEGMENT PAGE	Erzwingt Segmentbeginn an Seitengrenze (256 Bytes)
PARA	CODE SEGMENT PARA	Erzwingt Segmentbeginn an Paragraphgrenze (16 Bytes)
PURGE	PURGE PRINT	Makro-Definition wieder aufheben
RADIX	RADIX 16	Setzt Default-Zahlenbasis, so daß z.B. H entfällt
RECORD	C RECORD A:2,B:14	Weist A und B bestimmten Bits in C zu (s. a. MASK)
SEG	MOV AX,SEG XYZ	Liefert Basisadresse des log. Segments einer Variable
SEGMENT	STACK SEGMENT	Start einer Segment-Deklaration, auch CODE, DATA usw.
STACK	S1 SEGMENT STACK	Benennt das Stack-Segment (u.a. für Linker)
START:	START: PUSH CS	Standard-Label zur Markierung des Programmbeginns
STRUC	TAB1 STRUC	Definiert eine Tabellen-Struktur
SUBTIL	SUBTIL TEST2	Unter-Überschrift innerhalb eines Sourcefiles
THIS	BUF THIS BYTE	Bezug auf aktuelle Adresse
WORD	LB2 LABEL WORD	Kennzeichnet Wort (vgl. BYTE)

Das Source-Listing in *Abb. 3.1.3.4* enthält ein Beispiel-Programm, das für den Assembler MASM-86 erstellt wurde. Es klinkt sich in die Tastaturtreiber-Routine von MS-DOS-Rechnern ein, verbleibt dann zum großen Teil resident im Speicher und gestattet es so jederzeit, einen Epson-kompatiblen Drucker mit bestimmten Tastenkombinationen (ALT und Buchstabentaste) in nahezu jede gewünschte Druckart umzuschalten — etwas recht Nützliches also.

Die ORG-Anweisung am Programmbeginn sorgt dafür, daß der Code ab 100H beginnt, wie es für ein .COM-File nötig ist. Deshalb ist nach dem Linker-Aufruf (LINK, siehe MS-DOS) noch die Anwendung von EXE2BIN (ebenfalls im Kapitel über MS-DOS besprochen) erforderlich. Das fertige Programm sollte unmittelbar nach dem nationalen Tastatur-Treiber (z.B. nach KEYBGR.COM) geladen werden.

Nun aber zur Funktionsweise. Beim Laden erfolgt sofort ein Sprung zum Label INIT. Dort wird mit Hilfe der INT-21H-Funktion des Betriebssystems MS-DOS

```
;Druckersteuerung (c)H.F.
lpt = 0          ;Drucker-Nr.1
code segment
    org 100h
assume cs:code,ds:code,es:code
start:jmp init
newint:sti
    push ax
    push bx
    push cx
    push di
    push dx
    push ds
    push es
    mov ax,cs    ;DS=ES=CS
    mov ds,ax
    mov es,ax
    in al,60h    ;Scan-Code
    mov bl,al    ;-> BL
    and al,7fh
    mov cx,4
    mov di,offset spkeys
    cld          ;CTRL,ALT,
    repnz scasb ;Shift?
    jne notsp
    mov al,1     ;Bitmuster
    shl al,cl    ;schieben
    test bl,80h ;break?
    jz spmake
    not al       ;ja:Bit=0
    and flag,al
    jmp short intret
spmake:or flag,al;Make-Code
    jmp short intret
notsp:cmp byte ptr flag,8
    jne intret  ;nicht ALT
    mov ah,2     ;Drucker
    mov dx,lpt   ;überhaupt
    int 17h      ;bereit?
    and ah,0a8h
    cmp ah,80h
    jne intret  ;nein!
    mov al,bl    ;Scanc.->AL
    mov bx,offset tab
srch:mov ah,[bx]
    test ah,ah  ;Tab.-Ende?
    jz intret   ;ja,war nix
    cmp al,ah
    je found
    add bx,4     ;Je 4 Bytes
    jmp short srch
found:mov cx,3  ;noch drei...
print:inc bx
    mov al,[bx]
    cmp al,255  ;FFH=Ende
    je forget
    mov ah,0     ;DX noch
    int 17h      ;=LPT!
    loop print
    jmp short forget
intret:pop es
    pop ds
    pop dx
    pop di
    pop cx
    pop bx       ;jetzt zu
    pop ax       ;KEYBGR o.ä.
    db 0eah      ;Opcode JMP FAR
oldint:dd 0
forget:in al,61h;Keyboard
    or al,80h    ;rücksetzen
    out 61h,al
    and al,7fh   ;Taste
    out 61h,al   ;vergessen,
    mov al,61h   ;SEOI f.PIC
    out 20h,al
    pop es
    pop ds       ;nicht zum
    pop dx       ;Keyboard-
    pop di       ;Treiber
    pop cx
    pop bx
    pop ax
    iret

flag db 0  ;ALT/Shift/CTRL-Bits
spkeys db 38h,1dh,2ah,36h
tab:db 31h,27,120,1   ;N=NLQ
    db 18h,27,120,0   ;O=NLQ off
    db 13h,27,64,255 ;R=Reset
    db 26h,27,108,10 ;L=L.Margin 10
    db 32h,27,108,0   ;M=L.Margin 0
    db 19h,27,80,255 ;P=Pica
    db 12h,27,77,255 ;E=Elite
    db 1fh,27,109,255;S=Schmalschr.
    db 17h,27,52,255 ;I=Italics
    db 24h,27,53,255 ;J=Italics off
    db 11h,27,87,1    ;W=Dbl.Width
    db 2dh,27,87,0    ;X=Dbl.Wi.off
    db 2eh,15,255,0   ;C=Condensed
    db 20h,18,255,0   ;D=Cond.off
    db 21h,27,69,255 ;F=Fettschrift
    db 22h,27,70,255 ;G=Fettschr.aus
    db 16h,27,45,1    ;U=Underline
    db 2fh,27,45,0    ;V=Underl.off
    db 1eh,27,112,1   ;A=Proportional
    db 30h,27,112,0   ;B=proport.off
    db 0          ;00=Tabellenende

text db 13,10,'SETPRINT V 1.0$'
init:mov ax,3509h ;alten INT09h
    int 21h      ;merken
    mov word ptr oldint,bx
    mov ax,es
    mov word ptr oldint[2],ax
    mov dx,offset newint
    mov ax,2509h
    int 21h      ;neuer INT09H
    mov dx,offset text
    mov ah,9     ;Textmeldung
    int 21h
    mov dx,offset text
    int 27h      ;resident!
code ends
    end start
```

Abb. 3.1.3.4
Druckersteuerung mit ALT-Tastenkombinationen für MS-DOS-Rechner

der alte Tastatur-Interrupt-Vektor INT 09H unmittelbar hinter den Opcode eines Sprungbefehls (JMP FAR) gespeichert, so daß die neue Interrupt-Routine (NEWINT) nach Beendigung zum Tastatur-Treiber springen kann. Dann wird NEWINT ebenfalls mit Hilfe der MS-DOS-Funktion INT 21H als neuer Tastatur-Interrupt-Vektor für INT 09H eingesetzt. Schließlich deklariert ein INT 27H den Programmteil bis vor dem Label TEXT für resident und beendet das Programm. Das bedeutet, daß dem Betriebssystem der Speicherplatz, den die neue Interrupt-Routine sowie die zugehörige Druckercode-Tabelle belegen, als belegt gemeldet wird. Ein später geladenes anderes Programm wird also ab dem Label TEXT stehen.

Die neue Interrupt-Routine NEWINT rettet zunächst alle von ihr selbst verwendeten Register auf den Stack. Dann liest sie den Code der jeweiligen Taste von der (bei allen IBM-PC- und AT-kompatiblen Rechnern gleichen) I/O-Adresse 60H. Ist hierbei Bit 7 gesetzt, so bedeutet das, daß die Taste gerade losgelassen wurde (Break-Code), andernfalls wurde sie gedrückt (Make-Code). Die übrigen sieben Bits enthalten einen Scan-Code (Abtast-Code), der nur die Position der Taste auf dem Tastenfeld angibt; die später aufgerufene Tastaturtreiber-Interrupt-Routine (z.B. KEYBGR.COM) macht erst hieraus den „normalen" Zeichencode.

Mit Hilfe der Tabelle SPKEYS stellt die Routine NEWINT nun fest, ob die gedrückte oder losgelassene Taste ALT, CTRL oder eine der beiden Shift-Tasten war, und merkt sich das Ergebnis als Bitposition in der Speicherzelle FLAG. Dadurch kann festgestellt werden, welche Kombination von ALT, CTRL und Shift gerade aktiv ist — denn Druckercodes sollen nur dann gesendet werden, wenn ALT (und nur ALT) zusammen mit einer Buchstabentaste gedrückt wird. Wenn eine andere Sondertastenkombination aktiv ist, z.B. CTRL und ALT oder Shift, so erfolgt ein Rücksprung zur „alten" Interrupt-Routine, also z.B. zu KEYBGR.COM.

Der ROM-BIOS-Aufruf INT 17H (Register AH=2) dient nun dazu, erst einmal festzustellen, ob der Drucker (LPT1) überhaupt bereit ist. Wenn das nicht der Fall ist, erfolgt ebenfalls ein Rücksprung. Andernfalls wird die Tabelle TAB nach der außer ALT noch gedrückten Taste durchsucht. Jeder Eintrag in TAB besteht aus vier Bytes; das erste ist der Scan-Code der jeweiligen Buchstabentaste, die der Druckerfunktion zugeordnet wird. Die übrigen drei Bytes werden an den Drucker gesendet, wenn eben diese Taste zusammen mit ALT gedrückt wird. Sollen weniger als drei Zeichen ausgegeben werden, so kann der Rest des Eintrags mit 255 (= 0FFH) aufgefüllt werden. Das Ende der Tabelle ist mit einem Null-Byte markiert.

Wenn der Scan-Code in der Tabelle gefunden wurde, erfolgt die Ausgabe der restlichen drei Bytes des zugehörigen Tabelleneintrags an den Drucker. Anschließend springt das Programm zum Label FORGET, das den Tastatur-Interrupt ohne nachfolgenden Aufruf der früheren Interrupt-Routine beendet. Dadurch wird vermieden, daß die zur Druckersteuerung benutzten ALT-Tastencodes an ein Anwenderprogramm weitergereicht werden und dort vielleicht unerwünschte Effekte er-

zeugen. Das Ende des Hardware-Interrupts IRQ1, der dem Vektor von INT 09H entspricht, muß dem Interrupt-Controller 8259 an der Adresse 20H durch Übergabe des Steuerworts 61H mitgeteilt werden (SEOI = Specific End of Interrupt).

Das kleine Programm eignet sich z. B. bestens dazu, um dafür zu sorgen, daß ein an den Drucker ausgegebenes Disk-Directory-Listing (zu erzeugen mit DIR > PRN) mit einem zehn Zeichen breiten linken Rand zu Papier gebracht wird: Man braucht nur vorher ALT-L zu drücken. Die meisten Buchstaben sind so gewählt, daß eine Funktion mit dem nachfolgenden Buchstaben des Alphabets wieder ausgeschaltet werden kann; mit ALT-M wird der linke Rand also wieder auf Null gesetzt.

Der CP/M-Assembler ASM

Bei dem Disketten-Betriebssystem CP/M, das für die Prozessoren 8080/Z80, 8088/8086 und 68008/68000 erhältlich ist, wird ein Assembler und ein passender Editor mitgeliefert. Die Assembler für die verschiedenen Prozessoren unterscheiden sich im wesentlichen nur im Prozessor-Befehlssatz selbst, nicht in den Pseudobefehlen und in der Kommando-Syntax. Es genügt daher, wenn im folgenden auf den CP/M-80-Assembler bezug genommen wird. Er versteht allerdings nur die 8080-, nicht aber die zusätzlichen Z80-Befehle. Das Betriebssystem CP/M-80 selbst, von dem noch näher die Rede sein wird, ist nur mit 8080-Befehlen geschrieben und unverändert auf Z80-Computern lauffähig.

Zum Erstellen einer Quelltext-Datei wird bei CP/M der zeilenorientierte Editor ED mitgeliefert. Der Aufruf erfolgt mit ED XXX, wobei XXX der Name der zu erzeugenden oder zu ändernden Textdatei ist. Dem Benutzer stehen folgende Kommandos zur Verfügung (die Zahlenwerte sind als Beispiele aufzufassen):

5A	5 Zeilen aus der Originaldatei an den RAM-Textpuffer anhängen
OA	Zeilen aus der Originaldatei anhängen, bis Puffer halb voll
#A	Zeilen aus der Originaldatei anhängen, bis Puffer voll
B	Zeiger auf Pufferanfang stellen
−B	Zeiger auf das Pufferende stellen
−3C	Zeiger drei Zeichen rückwärts
3C	Zeiger drei Zeichen vorwärts
7D	7 Zeichen vor (−7D: hinter) dem Zeiger löschen
E	Datei auf Floppy retten und zurück zu CP/M
Fabc	Suchen der Zeichenkette abc, muß mit CTRL-Z enden

H	Datei auf Floppy retten und Editieren von vorn
I	Eingabemodus; Ende mit CTRL-Z
Iabc	Einfügen der Zeichenkette abc (CTRL-Z) an der Zeigerposition
6K	6 Zeilen ab (oder −6K: vor) dem Zeiger löschen
3L	Zeiger um 3 Zeilen nach unten (−3L: nach oben)
3M	Die nach M folgenden Kommandos dreimal ausführen
10:	Zeiger auf Zeile 10 stellen
5	Zeiger 5 Zeilen vorwärts und Zeile ausgeben
−5	Zeiger 5 Zeilen rückwärts und Zeile ausgeben
:10	Nachfolgendes Kommando nur bis Zeile 10 ausführen
O	Zur Originaldatei zurück
10P	10 Zeilen ab (oder −10P: vor) Zeiger ausgeben, Zeiger weiter
Q	Neu erstellte Datei löschen, Rückkehr zu CP/M (Vorsicht!)
R XYZ	Lesen der Datei XYZ.LIB in den Puffer
S abc CTRL-Z def CTRL-Z	Ersetzen von def durch abc
7T	7 Zeilen ausgeben, ohne den Zeiger zu verstellen
U	Umsetzung in Großbuchstaben
V	Zeilen-Numerierung ein- und ausschalten
OV	Anzeige des freien Speicherplatzes
8W	8 Zeilen auf die neue Datei ausgeben
8X	8 Zeilen auf eine LIB-Datei ausgeben
OX	Löschen der LIB-Datei
10Z	10 Sekunden Pause

Bei der Eingabe sind einige CTRL-Tastenkombinationen und die DEL-Taste (hex 7F) zu Steuerungszwecken zulässig:

CTRL-L	Zeilenende-Ersatz in Zeichenketten (z.B. in Suchstring)
CTRL-X	Aktuelle Zeile löschen
CTRL-Z	Zeichenketten-Trennzeichen bzw. Textende
CTRL-C	Verlassen einer Funktion
DEL	Ein Zeichen löschen

Der CP/M-Editor ED ist nicht sehr komfortabel. ASM-Quelltext-Dateien können aber auch mit anderen Textverarbeitungs.Programmen wie z.B. Wordstar erstellt werden. Die Assemblierung erfolgt bei ASM stets von Diskette auf Diskette, d.h. lediglich die Symboltabelle muß Platz im RAM haben. Der Assembler erzeugt aus der Quelltext-Datei (Name beispielsweise PGM.ASM) eine Objektcode-Datei PGM.HEX im Intel-Hex-Format (siehe Kapitel über Lochstreifen-Formate!) und

eine Textdatei PGM.PRN, die das Assemblerlisting inklusive Symboltabelle in druckreifem Format enthält.

Der Aufruf des Assemblers erfolgt unter CP/M durch Eingabe von ASM NAME.XYZ, wobei NAME der Name der Quelltext-Datei ist, X, Y und Z können jeweils folgende Werte annehmen:

A, B, ... P	Laufwerks-Name	Z	Datei nicht erzeugen
X	Ausgabe auf Bildschirm		

Ein Beispiel: Das Kommando ASM PGM.ABX holt den Sourcetext PGM.ASM von Laufwerk A, schreibt den Objektcode im Intel-Hex-Format als PGM.HEX auf Laufwerk B und gibt das Assemblerlisting nicht als PGM.PRN auf Disk, sondern nur auf den Bildschirm aus.

Doch nun zur Syntax des ASM-Quelltextes. ASM akzeptiert Namen für Labels (Symbole) bis zu 16 Zeichen Länge; das erste Zeichen muß dabei ein Buchstabe sein. Ein Dollar-Zeichen ist nicht signifikant und wird in Labels ignoriert (z.B. Ersatz für Leerstellen: LANGES$SYMBOL). Kommentare müssen mit einem Semicolon beginnen; steht dagegen schon als erstes Zeichen in einer Zeile ein Stern, so wird die ganze Zeile als Kommentar betrachtet. Außerhalb von Anführungszeichen werden alle Kleinbuchstaben in Großbuchstaben umgewandelt.

Numerische Konstanten können in vier Zahlensystemen oder als Zeichenkette eingegeben werden, indem man der Zahl einen Buchstaben zur Kennzeichnung folgen läßt bzw. ASCII-Zeichen mit Anführungszeichen umschließt. Auch die vier Grundrechenarten (+ − * /) sowie logische Verknüpfungen sind erlaubt:

1010B	Binäre Konstante (Dualzahl)
65O oder 65Q	Oktale Konstante (O wie Otto, nicht Null!)
96D	Dezimale Konstante 96
0F9H	Hexadezimale (sedezimale) Konstante F9
'A'	ASCII-Zeichen A
'ABcd'	Zeichenkette ABcd
0FFH − 34	Hex FF minus dezimal 34
A AND B	A und B Und-verknüpft
A OR B	A und B Oder-verknüpft
A XOR B	Bitweise Exklusiv-Oder-Verknüpfung
A SHL B	A um B Bits nach links schieben
A SHR B	A um B Bits nach rechts schieben
NOT A	Alle Bits von A invertieren
A MOD B	Rest der Division A/B

Wie in Labels dürfen auch hier Dollar-Zeichen als nicht signifikante Trennzeichen verwendet werden, um z.B. die Lesbarkeit einer langen Dualzahl zu verbessern:

0100$0110$1010$0001 ist gleichwertig mit 0100011010100001, aber wegen der Unterteilung in 4-Bit-Gruppen wesentlich besser lesbar. Ein alleinstehendes Dollar-Zeichen hat dagegen sehr wohl eine Bedeutung, es liefert nämlich den aktuellen Stand des Programmzählers beim Assemblieren.

Neben den 8080-Mnemonics besitzt der CP/M-80-Assembler noch eine Reihe von Pseudo-Befehlen, die lediglich Steuerungszwecken dienen und keinen Objektcode erzeugen:

ORG	Anfangsadresse setzen
END	Ende der Assemblierung (kann entfallen)
EQU	Einmalige Wertzuweisung
IF	Zeilen bis ENDIF nur dann assemblieren, wenn der nach IF folgende Ausdruck ungleich Null ist
ENDIF	Ende des bedingt zu assemblierenden Quelltextes
DB	Datenbyte(s) ablegen
DW	Datenworte (16 Bit) ablegen, niederwertige Byte zuerst
DS	Speicherplatz reservieren

ASM gibt eventuelle Fehlermeldungen außer bei Disk-Errors nicht im Klartext, sondern als einen einzigen Buchstaben aus:

Buchstabe	Englisch	Deutsch
D	Data Error	DB/DW-Anweisung fehlerhaft
E	Expression	Ausdruck kann nicht ausgewertet werden
L	Label	Symbol fehlerhaft, z.B. doppelt definiert
N	Not implemented	Unzulässiger Pseudobefehl
O	Overflow	Wert eines Ausdrucks unzulässig
P	Phase Error	Symbol wurde umdefiniert
R	Register	Register-Angabe unvollständig
S	Syntax	Falsche Anweisung
U	Undefined Label	Symbol wurde nicht definiert
V	Value	Fehlerhafter Operand

Eine mit ASM erzeugte Intel-Hex-Datei PGM.HEX kann mit dem CP/M-Befehl LOAD PGM in eine „echte" Objektcode-Datei PGM.COM umgewandelt werden, die man später einfach durch Eintippen von PGM laden und starten kann. Dazu mehr im Kapitel über das Betriebssystem CP/M.

3.2 Höhere Programmiersprachen

3.2.1 Interpreter

Wer schon einmal einen Heimcomputer programmiert hat, der hat das vermutlich zunächst einmal nicht in Maschinensprache oder mit Hilfe eines Assemblers getan, sondern gewöhnlich mit einem Basic-Interpreter. Basic ist (neben anderen Sprachen) eine sogenannte höhere Programmiersprache.

Eine höhere Programmiersprache ist im Gegensatz zu Maschinensprache bzw. Assembler nicht maschinen-, sondern problemorientiert. Das bedeutet, daß ihr Befehlssatz sich weniger am Hardware-Aufbau bzw. Prozessortyp des Computers orientiert, sondern so zugeschnitten ist, daß sich typische Computer-Anwendungen möglichst einfach, zeitsparend und überschaubar programmieren lassen.

Ein Befehlssatz, der von dem des Prozessors selbst abweicht, muß vom Computer natürlich irgendwie simuliert werden. Wenn das z. B. in Basic geschriebene Programm im Speicher etwa so wie ein Assembler-Quelltext abgelegt ist, also so, wie es der Benutzer eingegeben hat, so kann der Prozessor die darin enthaltenen Befehle nicht direkt ausführen.

Ein Interpreter ist nun ein Programm, das für jedes von ihm im Quelltext gefundene Befehlswort eine in Maschinensprache geschriebene Routine enthält, die genau das tut, was der Programmierer mit dem jeweiligen Befehlswort ausdrücken wollte.

Ein Beispiel: Bei dem Basic-Befehl PRINT"WUNDERBAR" sucht der Interpreter zunächst in seiner Befehls-Tabelle das Wort PRINT. Sobald er es gefunden hat, ermittelt er aus einer weiteren Tabelle die Adresse des Unterprogramms, das diesen Befehl ausführt. Dieses Unterprogramm stellt fest, daß das auf PRINT folgende Zeichen ein Anführungszeichen ist und schließt daraus, daß er den von Anführungszeichen eingeschlossenen Text auf den Bildschirm ausgeben soll. Sobald das geschehen ist, erfolgt die Rückkehr aus dem PRINT-Unterprogramm zur Interpreter-Hauptschleife, die den nächsten Befehl decodiert, wieder das zugehörige Unterprogramm aufruft und so weiter.

Tokens als Befehls-Kürzel

Die meisten heute üblichen Interpreter speichern die vom Benutzer eingegebenen Programme allerdings nicht als Folge von ASCII-Zeichen, sondern wandeln jedes Befehlswort in ein einziges Byte um, das sogenannte Token. Dadurch ist zum Beispiel für den Basic-Befehl PRINT nur ein einziges Byte statt fünf nötig. Damit der Interpreter später, wenn das Programm z. B. mit RUN gestartet wird, weiß, was im Quelltext ASCII-Zeichen (Text, Variablen usw.) und was Tokens sind, werden alle Tokens gekennzeichnet, indem ihr höchstwertiges Bit 1 ist; bei ASCII-Zeichen ist es dagegen immer Null. Neben der Speicherplatz-Ersparnis hat dieses Token-Verfahren auch den Vorteil, daß der Interpreter während des eigentlichen Programm-

laufs nicht mühsam reservierte Worte in einer Tabelle suchen muß; vielmehr verwendet er das Token selbst als Index zu einer Tabelle, in der die Adressen der zugehörigen Unterprogramme stehen.

Die meisten Interpreter (z.B. die Microsoft-Basic-Dialekte) lassen bei der Eingabe Zahlenwerte als ASCII-Zeichenfolgen im Programmtext stehen. Besonders schnelle Interpreter wandeln sie dagegen schon in der Eingabephase in die interne, binäre Zahlendarstellung um, so daß während des Programmlaufs diese Umwandlung nicht mehr nötig ist (z.B. DAI-Basic).

Zeilen-Zeiger (Link)

Üblich ist dagegen wiederum die Methode, am Anfang jeder Zeile nicht nur die Zeilennummer, sondern auch die Anfangsadresse der nächsten Zeile im Speicher festzuhalten. Das geschieht in zwei Bytes, die man englisch auch Link nennt (Verbinder). Dem Interpreter fällt es so nämlich leichter, das Programm z.B. bei der Ausführung einer GOTO-Anweisung (Sprungbefehl) nach einer bestimmten Zeilennummer zu durchsuchen oder, z.B. nach einer IF-Anweisung (bedingter Sprung) bei der nächsten Zeile fortzufahren. Bei Microsoft-Basic-Interpretern und ihren zahlreichen Abkömmlingen (Commodore, Apple, AIM-65, CP/M-MBasic u.a.) erfolgt die Darstellung der Zeile

10 PRINT "?"

in folgendem Format, wobei die Zeilennummer 10 als 2-Byte-Binärzahl (Low/High Byte) gespeichert wird:

Adresse	Inhalt	Bedeutung
0211	00	Markierung des Programmanfangs
0212	1B	Niederw. Teil der Adresse der nächsten Zeile
0213	02	Höherw. Teil der Adresse der nächsten Zeile
0214	0A	Niederw. Teil der Zeilennummer (10)
0215	00	Höherw. Teil der Zeilennummer
0216	97	Token für PRINT-Anweisung
0217	22	Anführungszeichen in ASCII
0218	3F	Fragezeichen in ASCII
0219	22	Anführungszeichen in ASCII
021A	00	Markierung für Ende der Zeile
021B	00 00	Zwei Null-Bytes statt der Adresse der nächsten Zeile markieren das Programmende

Die hier genannten Adressen und der Token-Wert für PRINT gelten für das AIM-65-Basic; der prinzipielle Aufbau einer Basic-Zeile ist jedoch bei anderen Computern genauso. So ist z.B. bei Apple-II und C-64 die Anfangsadresse nicht hex 0211, sondern hex 800.

Numerische Variablen

In den meisten höheren Programmiersprachen kann man Zahlenwerte und Texte sogenannten Variablen zuweisen, z.B. A=13. Meist dürfen Variablennamen aus ein oder zwei Zeichen bestehen, wobei das erste ein Buchstabe und das zweite ein Buchstabe oder eine Ziffer sein muß. Ist der Name länger, so werden oft trotzdem nur die ersten beiden Zeichen verwertet, je nach Interpreter.

Der Interpreter legt numerische Variablen nach dem Programmstart ab der Endadresse des eigentlichen Programms ab. Bei Apple-II, AIM-65, C-64 und ähnlichen Computern belegt jede solche Variable sieben Bytes: zwei für den Variablennamen und fünf für den Wert. Fließkomma-Variablen, die durch Verwendung eines Zehnerexponenten (z.B. 2.7E-22) einen großen Wertebereich darstellen können, nutzen diese fünf Bytes voll aus, während bei Ganzzahl-Variablen nur zwei verwendet werden und die übrigen drei ungenutzt bleiben.

Die Zahlendarstellung erfolgt entweder im BCD-Format oder binär. Das BCD-Format spaltet z.B. den Ganzzahl-Wert 32767 so auf drei Hex-Bytes auf:

32767 = 03 27 67

Berechnungen können nun z.B. beim Prozessor 6502 unter Zuhilfenahme der BCD-Arithmetik (Binary Coded Decimal; 6502-Befehl SED) quasi dezimal, also ohne Umrechnung in eine Dualzahl, durchgeführt werden. Allerdings ist insbesondere bei Multiplikation und Division eine BCD-Arithmetik wesentlich langsamer als eine auf Dualzahlen beruhende. An der Beispiel-Zahl 32767 läßt sich aber auch zeigen, daß sie weniger Speicherplatz benötigt:

32767 = hex 7F FF

Das ist gleichzeitig die mit 15 Bits maximal darstellbare Zahl (2↑15). Das 16. Bit der beiden Bytes verwendet man sinnvollerweise als Vorzeichen-Bit; ist es Null, so handelt es sich um eine positive Zahl, andernfalls um eine negative.

Die meisten Interpreter stellen eine Integer-Variable (Ganzzahl) tatsächlich in diesem 2-Byte-Format dar und lassen dafür somit den Wertebereich −32768... +32767 zu.

Bei Fließkomma-Variablen wird außer der Mantisse auch ein Exponent mitgespeichert. Bei den meisten Microsoft-Basic-Dialekten erfolgt die interne Darstellung solcher Zahlen in fünf Bytes, und zwar so normiert, daß das Komma stets nach der vordersten Ziffer, im Dualsystem also nach der vordersten Eins steht. Zu diesem Zweck wird der Exponent entsprechend justiert. Dazu wird solange durch 2 dividiert bzw. mit zwei multipliziert, bis das Ergebnis eine Zahl $1 <= p <= 2$ ist. Die Anzahl der Divisionen bzw. Multiplikationen ist vom Exponent abzuziehen oder ihm hinzuzuzählen.

Eines der fünf Bytes speichert den Exponenten mit seinem Vorzeichen. Die übrigen vier enthalten die normierte Mantisse, wobei das höchstwertige Bit ihr Vorzeichen repräsentiert.

Eine gewisse Problematik der intern dualen Zahlendarstellung ergibt sich beim Versuch, eine Zahl zu runden bzw. in Basic mit INT den Nachkommateil abzutrennen. Für positive Zahlen verschiebt INT das Komma der Mantisse entsprechend dem Zweier-Exponenten und setzt alle Bits hinter dem Komma Null. Wenn die entstandene Zahl nicht ohnehin Null ist, wird nun das Komma wieder solange nach links verschoben, bis nur noch eine Dualzahl vor dem Komma steht. Die Anzahl der Schiebe-Operationen ergibt dann wieder den Exponenten. Die Mantisse wird hierbei also nicht gerundet, sondern lediglich verkürzt. Auch bei negativen Zahlen wird die nächstniedrigere Zahl geliefert: INT(1.8)=1, INT(−1.8)=−2! Diese Tatsache kann zu recht merkwürdigen Ergebnissen führen. Die folgenden drei Programmbeispiele, die rein arithmetisch immer das gleiche Ergebnis bringen müßten, machen dies deutlich:

```
10  S=66/20
20  E=INT(10*S)/10
30  PRINT E
```

Ergebnis: 3.2

```
10  S=66/20
20  S=S*10
30  E=INT(E)
40  E=E/10
50  PRINT E
```

Ergebnis: 3.3

```
10  S=66/20
20  E=INT(10*S*1)/10
30  PRINT E
```

Ergebnis: 3.3

Im dritten Fall wird eine korrekte Rundung erzwungen, indem nach S*10 nochmals mit 1 multipliziert wird.

String-Variablen

Bei Variablen, die Zeichenketten darstellen (Stringvariable, in Basic z.B. A$), schreibt der Interpreter in die sieben Bytes wiederum zwei Zeichen für den Namen in den Speicher, jedoch dann nicht gleich die Zeichenkette selbst, sondern ein Byte für deren Länge und zwei für ihre Anfangsadresse. Die übrigen zwei der insgesamt wieder sieben Bytes dieses „String-Deskriptors" bleiben unbenutzt. Wird die Stringvariable im Programm selbst definiert, so zeigt der „Pointer" in der Variable in den Programmtext. Am Beispiel des AIM-65 sei dies verdeutlicht:

```
10  A$="ABC"
```

Dieses Mini-Basic-Programm und die Variable A$ finden sich nach RUN (erst dann wird ja die Variable wirklich angelegt!) so im Speicher wieder:

```
0211  00              Programmanfangs-Markierung
0212  1F  02          Adresse der nächsten Zeile (Low/High)
0214  0A  00          Zeilennummer 10 (Low/High)
0216  41  24  AC  22  A$=" in ASCII
021A  41  42  43      ABC  in ASCII
021D  22              "    in ASCII
021E  00              Zeilenende-Markierung
021F  00  00          Programmende-Markierung
```

```
0221  41  80        Variablenname; Bit 7 = 1 in Byte 2: Stringvariable
0223  03           Länge des Strings ABC
0224  1A  02        Anfangsadresse des Strings ABC
0226  00  00        Nicht verwendet
```

Wird dagegen ein String von der Tastatur eingegeben oder innerhalb des Programms aus anderen kombiniert, so legt ihn der Interpreter von „oben" herab (also ab der höchsten zur Verfügung stehenden RAM-Adresse) nach unten im Speicher ab. Ein kleines Experiment macht dies sichtbar:

```
10  A$="ABC"+"D"
```

Hier wird der String ABCD aus zwei Strings zusammengefügt. Dadurch ist der Interpreter gezwungen, die so entstandene Zeichenkette am oberen RAM-Ende abzulegen. Im Speicher eines mit 4 KByte RAM ausgestatteten AIM-65 (RAM-Ende = 0FFF) findet man nach RUN folgendes:

```
0211  00              Programmanfangs-Markierung
0212  23  02 0A 00    Startadresse der nächsten Zeile, Zeilennummer 10
0216  41  24          A$     in ASCII
0218  AC              Token für "="
0219  22              Token in ASCII
021A  41  42 43       ABC  in ASCII
021D  22              ABC  in ASCII
021E  A4              Token für "+"
021F  22  44 22       "D"    in ASCII
0222  00  00 00       Zeilen- und Programmende-Markierungen
0225  41  80          Variablenname A$
0227  04              Länge der Variable
0228  FC  0F          Anfangsadresse der Variable (Low/High)

0FFC  41  42 43 44    ABCD in ASCII
```

Im Lauf eines Programms, das viele String-Operationen enthält, werden natürlich immer neue Strings angelegt; sie wachsen also im RAM von „oben" nach „unten". Gleichzeitig wachsen numerische Variable sowie die String-Deskriptoren vom Programmende nach oben. Irgendwann ist dann kein Platz mehr für zusätzliche Strings. Der Interpreter führt dann eine sogenannte „Garbage Collection" durch, eine „Müll-Aufsammlung". Er kontrolliert nämlich, ob die früher angelegten Strings am oberen RAM-Ende überhaupt noch gebraucht werden. Das geht leicht anhand der String-Deskriptoren: Weist auf einen „oben" abgelegten String kein Deskriptor-Zeiger, so kann er gelöscht werden, so daß die tieferliegenden nach oben nachrücken können. Dadurch wird wieder Platz im RAM frei. Diese Garbage Collection kann je nach Speicherumfang einige Sekunden, ja Minuten dauern.

Nicht alle Basic-Interpreter arbeiten mit dieser „dynamischen" Vergabe von String-Speicherplatz. Bei manchen Computern (z.B. HP-80-Serie) muß man am Programmanfang definieren, wie lang Zeichenketten maximal sein dürfen. Jede

Stringvariable erhält dann vom Interpreter einen bestimmten Speicherplatz zugewiesen, der nur für sie allein da ist. Eine Garbage Collection ist dann nicht notwendig. Allerdings wird u. U. eine erhebliche Menge an Speicherplatz vergeudet, wenn die angelegten Strings im Mittel wesentlich kürzer sind als die am Programmanfang festgelegte Maximallänge.

Arrays (Feldvariablen)

Die meisten Interpreter lassen auch sogenannte Arrays als Variablentyp zu. Darunter versteht man ein ganzes Feld einzelner Werte, das mit einem einzigen gemeinsamen Variablennamen gekennzeichnet ist. Der Zugriff auf die einzelnen Werte erfolgt mit einer Zahl als Index, die gewöhnlich 0...65536 (= 2↑16) betragen darf. Der maximal zulässige Index muß am Programmanfang definiert werden; in Basic erfolgt das mit dem DIM-Befehl, z. B.

```
DIM A%(4),A$(100),B(20)
```

Fehlt eine DIM-Anweisung (Dimensionierung) am Programmanfang, so nimmt der Interpreter beim ersten Auftreten der Index-Variablen im Programm einen Maximal-Index von 10 an.

Arrays benötigen sieben Bytes Speicherplatz für den Header (Variablen-Kopf) und darüber hinaus noch je nach Typ unterschiedlich viele Bytes für die eigentlichen Werte: bei Ganzzahl-Variablen (Integers) 2 Byte pro Wert, bei Fließkomma-Variablen 5, bei String-Variablen drei (nämlich eines für die Länge und zwei für die Anfangsadresse des zugehörigen Strings).

Vor- und Nachteile

Die interpretative Abarbeitung von Befehlen führt natürlich dazu, daß diese Programme wesentlich langsamer ablaufen als solche, die z. B. in Assembler geschrieben worden sind. Dieser Geschwindigkeits-Unterschied kann Faktor 100 und mehr sein. Für zeitkritische Aufgaben ist ein Interpreter daher wenig geeignet.

Ein großer Vorteil ist neben der Möglichkeit, in höheren, anwendungsfreundlicheren Programmiersprachen zu arbeiten, daß ein eingetipptes Programm sofort lauffähig ist (im Gegensatz zum Assembler, oder, siehe unten, zu Compilern) und sich immer unter der Kontrolle des Interpreters befindet, d. h. normalerweise nicht durch Fehleingaben oder falsche Programmierung in die Tiefen der Maschine „abstürzen" kann.

Ferner ist es bei Interpretern möglich, Befehle im Direkt-Modus einzugeben und so sofort ihre Wirkung zu testen. Dies geschieht einfach durch Weglassen der Nummer am Zeilenanfang. Diese Dialogfähigkeit macht die Programmierung zusammen mit den ausführlichen Klartext-Fehlermeldungen moderner Interpreter auch für Anfänger ohne große Erfahrung möglich.

3.2.2 Compiler

Bei einem Interpreter sind die auszuführenden Befehle der höheren Programmiersprache während der Programmausführung dauernd im Speicher und werden erst dann decodiert, wenn sie tatsächlich aufgerufen werden. Da der Decodier-Vorgang Zeit kostet, läßt sich mit diesem Prinzip keine allzu hohe Ausführungs-Geschwindigkeit erzielen. Auch die Verwendung von Tokens bringt nur eine geringfügige Verbesserung.

Compiler arbeiten dagegen ähnlich wie Assembler, nur daß der Quelltext hier nicht aus Assembler-Mnemonics, sondern aus den Befehlen einer höheren Programmiersprache besteht und ein Quelltext-Befehl gewöhnlich nicht nur einen einzigen Objektcode-Befehl, sondern mehrere erzeugt. Am ehesten kann man einen Compiler also mit einem Makro-Assembler vergleichen, wobei für jeden Befehl der Programmiersprache eine Assembler-Befehlsfolge als Makro vordefiniert ist.

Geschwindigkeit

Letztendlich erzeugt ein Compiler also ein Maschinenprogramm, das vom Prozessor deutlich schneller abgearbeitet werden kann als ein Interpreter-Programm. (Manchmal wird auch der Umweg über die Erzeugung von Assembler-Mnemonics als Zwischencode gemacht, die dann noch assembliert werden müssen.) Allerdings ist die Intelligenz eines Compilers im Vergleich zu der eines durchschnittlichen Assembler-Programmierers recht gering, und man darf daher nicht erwarten, daß ein compiliertes Programm, dessen Quelltext z.B. in Basic oder Pascal geschrieben wurde, mit einem direkt in Assembler geschriebenen Schritt halten kann. Denn dem Compiler fehlt die Fähigkeit, z.B. die Genauigkeit von Rechen-Routinen selbständig an die für das Problem wirklich nötige anzupassen und die Register der CPU effektiv z.B. als Zähler zu verwenden, statt dafür jedesmal eine Anzahl von Speicherzellen als Variable zu reservieren.

Je nach Problem und Compiler kann man damit rechnen, daß ein compiliertes Programm gegenüber einem interpretierten etwa um den Faktor 2...10 schneller abläuft; es ist damit aber immer noch mindestens 10mal langsamer als ein reines Assembler-Programm. Lediglich sehr rechenintensive Aufgaben, wenn z.B. häufig Sinus- oder Logarithmus-Werte errechnet werden müssen, oder dauernde Zugriffe auf externe Massenspeicher verringern dieses Verhältnis.

Während insbesondere die Sprache Basic, aber auch Logo, aus Gründen der Dialogfähigkeit vorwiegend mit Interpretern implementiert wird, sind für andere Sprachen wie Pascal, Fortran, Cobol, C und andere praktisch ausschließlich Compiler üblich. Der Grund ist vor allem, daß Sprachen mit komplexen Strukturen (WHILE...WEND, DO...LOOP, CASE usw.) bei interpretativer Abarbeitung zu sehr langsamen Programmen führen würden.

Compiler stellen Zahlen intern genauso dar wie Interpreter, nämlich entweder im BCD- oder im binären Format. Im erzeugten Objektcode (Run Time Code,

Laufzeit-Code) werden gewöhnlich allerdings die im Quelltext verwendeten Variablennamen nicht mehr verwendet, sondern (je nach Variablentyp) Gruppen von Speicherzellen, auf die durch Zeiger schnell zugegriffen werden kann.

Das Laufzeit-Paket

Außer dem erzeugten Objektcode ist noch ein gewisser „Overhead" im Speicher notwendig, das sogenannte „Laufzeit-Paket" (Run Time Unit). Es hängt vor allem von der Eigenintelligenz des Compilers ab, wie umfangreich es sein muß. Im ungünstigsten Fall enthält es u.a. alle Arithmetik-Unterprogramme als Assembler-Routinen, inklusive Sinus, Tangens, Cosinus, Logarithmus usw. Schlauere Compiler stellen dagegen während des Übersetzungs-Vorgangs fest, welche Funktionen überhaupt im Programm aufgerufen werden und bauen nur die dafür nötigen Unterprogramme in das Laufzeit-Paket mit ein.

In den meisten Fällen ist der erzeugte Code nur auf dem Computer lauffähig, auf dem er vom Compiler erzeugt wurde, da extensiver Gebrauch von System-Adressen gemacht wird. Wenn jedoch ein Programm für einen anderen Computer erzeugt werden soll (z.B. für einen Einplatinen-Computer), dann benötigt man einen sogenannten „Target Compiler". Er ermöglicht es, daß der Programmierer vor der Compilierung alle spezifischen Adressen des Zielsystems im Quelltext vordefiniert; sie müssen nicht mit jenen des Entwicklungssystems (des Rechners, auf dem der Compiler läuft) übereinstimmen.

Vor- und Nachteile

Es war davon die Rede, daß compilierte Programme deutlich schneller ablaufen als interpretierte. Je nach Laufzeit-Paket-Overhead kann darüber hinaus auch eine Speicherplatz-Ersparnis erwartet werden. Allerdings haben Compiler auch Nachteile: Wenn während der Laufzeit des erzeugten Programms ein Fehler auftritt, so wird dem Benutzer meist nicht die Zeilennummer des Quellprogramms gemeldet, die dem gerade ausgeführten (fehlerhaften) Objektcode entspricht — denn der Quellcode steht ja während des Programmlaufs gar nicht mehr im Speicher! Die Fehlersuche kann also u.U. problematischer als bei Interpretern sein.

Hinzu kommt, daß nach Programmänderungen vor dem Neustart das Quellprogramm erst wieder neu übersetzt werden muß. Schnell mal eben die Wirkung einer Änderung auszuprobieren, ist beim Compiler also umständlicher und zeitraubender als bei einem Interpreter, wenn sich auch z.B. beim Turbo-Pascal-Compiler Tendenzen zu größerer Benutzerfreundlichkeit abzeichnen.

Besonders praktisch ist es für Basic-Programmierer, daß manche Compiler denselben Quelltext praktisch ohne vorherige Anpassungen übersetzen können, der ursprünglich für den Interpreter geschrieben wurde (z.B. Speedstar oder TASC für das Applesoft-Interpreter-Basic).

3.2.3 Basic

Bei Mikrocomputern ist Basic nach wie vor die am weitesten verbreitete höhere Programmiersprache. „Basic" ist die Abkürzung von „Beginner's All-Purpose Symbolic Instruction Code", also etwa „Symbolischer Befehlssatz für Anfänger". In der Tat ist das Programmieren in Basic auch für Anfänger leicht und schnell erlernbar. Professionelle Informatiker werfen der Sprache allerdings oft vor, sie würde nicht genügend zum sauber strukturierten Programmieren erziehen, da man mit GOTO kreuz und quer im Programm hin- und herspringen kann und man schnell den Überblick verliert, welche Variablennamen man bei späteren Erweiterungen noch verwenden darf, da sie nicht lokal für einzelne Unterprogramme, sondern grundsätzlich für das gesamte Programm definiert sind (vgl. lokale Variable in Pascal).

Basic-Programme bestehen aus numerierten Zeilen. Sinnvollerweise erfolgt diese Numerierung nicht in fortlaufender Folge (1, 2, 3 usw.), sondern in gewissen Abständen 10, 20, 30 usw.), damit man bei späteren Programmänderungen noch Platz für das Einfügen von neuen Zeilennummern hat. Der Aufruf von Unterprogrammen mit GOSUB oder das Springen zu anderen Programmzeilen mit GOTO erfolgt im Ur-Basic stets mit solchen Zeilennummern. Nur wenige Basic-Interpreter und Compiler lassen ähnlich wie Assembler bzw. wie Pascal Namen dafür zu.

Leider gibt es „das" Basic schlechthin nicht. In den Jahren seit 1962, als John Kemeny und Thomas Kurtz einen Basic-Grundwortschatz entwickelten, haben sich zahlreiche untereinander nicht kompatible Dialekte dieser Sprache entwickelt. Das geht so weit, daß nicht einmal Befehle, die genau die gleiche Wirkung haben, den gleichen Namen haben müssen. Eine Art de-facto-Standard stellt jedoch die Syntax (Befehls-Schreibweise) der von der amerikanischen Firma Microsoft für unterschiedlichste Mikroprozessoren entwickelten Basic-Interpreter dar, so für die CPUs 6502, 8080/Z80, 8088/8086 und andere. Die Befehlssätze vieler marktüblicher Computer stellen davon eine Untermenge dar.

MBasic

Microsoft entwickelte zahlreiche Basic-Interpreter für unterschiedliche CPUs. Die für den 8080 geschriebene Version Basic-80 wird auch MBasic genannt und ist bei CP/M-Computern recht verbreitet. Der Befehlssatz ist recht umfangreich und wird im folgenden beschrieben, da er als beispielhaft gelten kann. Optionale Angaben stehen in spitzen Klammern. Variablen oder numerische Argumente müssen vom Befehl durch einen Leerraum getrennt werden.

Kommandos und Befehle

AUTO <n <,m>>
Automatische Zeilennumerierung ab n mit Inkrement m zur komfortableren Programmeingabe.

CALL Variable <,Argumentliste>
Aufruf eines Maschinenprogramms an der durch die Variable gekennzeichneten Adresse. Eine Parameter-Übergabe ist mit der Argumentliste möglich.

CHAIN N$
Programm namens N$ von Diskette laden und starten, ohne die Variableninhalte zu zerstören.

CLEAR <n> <,m>
Variablenwerte löschen, Disk-Files schließen, Stack zurücksetzen. n gibt optional die oberste RAM-Grenze für Basic an, m den Stack-Speicherplatz. Bei frühen MBasic-Versionen war n der String-Speicherplatz und m das RAM-Ende.

CLOSE <#> n <,m,...>
Datei mit Nummer n (bzw. m usw.) schließen.

COMMON A, B(), C$
Vor einem CHAIN-Befehl wird bewirkt, daß nicht alle Variablen vom neuen Programm übernommen werden, sondern nur die nach COMMON spezifizierten.

CONT
Nach Ausführung von STOP oder END oder nach Abbruch des Programms mit CTRL-C wird dieses beim nächsten Befehl fortgesetzt. Funktioniert nicht nach einer Fehlermeldung oder nach Programmänderungen.

DATA 1,2,A,",",0,ABC," XYZ"
Mit READ-Befehlen können die Werte nach DATA einmal in derselben Reihenfolge gelesen werden. Ein erneutes Lesen ist nur nach RESTORE möglich. Strings müssen nicht in Anführungszeichen stehen, es sei denn, sie enthalten Kommata, Doppelpunkte oder (am Anfang oder Ende) Leerräume.

DEFINT, DEFDBL, DEFSNG, DEFSTR V
Definition von Variablentypen für ganze Zahlen, Fließkommazahlen mit doppelter Genauigkeit, Fließkommazahlen mit einfacher Genauigkeit oder Strings. Die Variablen können auch als Liste (A, B, C) oder als Alphabets-Bereich (A—C) geschrieben werden.

DEF USR <n> = m
Zuweisen der Integer-Startadresse m für den zugehörigen Befehl USR mit optional n als Nummer der Maschinenroutine.

DELETE n<—m>
Löschen der Zeile mit der Nummer n oder der Zeilen n bis m.

DIM A(40),B$(20)
Dimensionieren von Arrays (Feldern), hier 40 Elemente für A und 20 für B$. Wenn ein Feld ohne vorherigen DIM-Befehl verwendet wird, nimmt der Interpreter 10 als Dimensionierung an.

EDIT n
Zeile n editieren. Hierfür stehen dann folgende Steuerbefehle zur Verfügung:

Leerraum	Cursor nach rechts
Delete	Cursor nach links
Return	Geänderte Zeile speichern
I	Text am Cursor einfügen
D	Zeichen unter dem Cursor löschen
nD	n Zeichen rechts vom Cursor löschen
H	Rest der Zeile ersetzen
X	Zeile verlängern
Sz	Suche nach dem Zeichen z
nSz	Suche nach dem n-ten Auftreten des Zeichens z
K	Wie S, aber die gefundenen Zeichen werden gelöscht
C	Zeichen unter dem Cursor ersetzen
nC	n Zeichen ab dem Cursor ersetzen
E	Wie Return, Rest der Zeile wird aber nicht angezeigt
Q	EDIT-Modus verlassen, Zeile unkorrigiert lassen
L	Zeile listen und Cursor an den Anfang
A	Editiervorgang von vorn beginnen

Nach Syntax-Fehlermeldungen gelangt man automatisch in den EDIT-Modus. Bei der normalen Programmeingabe gelangt man dorthin auch mit CTRL-A. Wenn man statt der Zeilennummer nach EDIT einen Punkt eingibt, kann man die zuletzt bearbeitete Zeile editieren.

END
Ende der Programmausführung; am Programmende kann END jedoch auch entfallen.

ERASE A,B,C
Die Array-Variablen A, B, C werden aus dem Speicher eliminiert und können mit DIM neu dimensioniert werden.

ERR
Nach Fehlermeldungen liefert ERR die Fehler-Nummer. Beispiel:

```
IF ERR=15 THEN PRINT"String zu lang!"
```

ERROR n
Anweisung, einen Fehler mit der Nummer n zu simulieren.

FIELD <#> n,m AS V$
Definieren von Speicherplatz im Puffer für Random-Disk-Zugriff mit der Variable V$ und n als File-Nummer mit der Feldlänge m.

FOR...TO...NEXT <...STEP>
Programmschleife mit FOR Variable=Anfangswert TO Endwert, NEXT <Variable> und optional STEP Inkrement/Dekrement. Beispiel:

```
FOR I=1 TO 10 STEP 0.1:PRINT SQR(I):NEXT
```

Diese Befehlsfolge druckt die Quadratwurzeln des Bereichs 1...10 in Schritten von 0,1, also hundert Werte. Wenn nach NEXT keine Variable steht, wird die beim letzten FOR-Befehl angegebene angenommen.

GOSUB n...RETURN
Mit GOSUB Zeilennummer kann man ein Unterprogramm aufrufen, das mit RETURN endet und damit zum Hauptprogramm zurückkehrt.

GOTO n
Sprungbefehl zur Zeilennummer n. Wenn es diese nicht gibt, erfolgt eine Fehlermeldung.

IF...THEN <...ELSE>
Bedingte Verzweigung. Nach IF steht ein logischer Vergleich oder eine boolesche Variable (0 = falsch, −1 = wahr!). Die Anweisungen nach THEN bis optional ELSE werden nur ausgeführt, wenn der Ausdruck nach IF wahr ist. Befehle nach ELSE bis zum Zeilenende werden dagegen nur bei unwahrem IF-Ausdruck ausgeführt. GOTO kann nach THEN und ELSE auch entfallen. Beispiele:

IF A=10 THEN PRINT"A=10" ELSE PRINT"A<>10"
IF A THEN PRINT"A IST NICHT NULL"
IF A$="JA" THEN 50 (oder THEN GOTO 50)

INPUT <"Text";>V

Eingabe eines Wertes für die Variable V (oder eine Variablenkette, z.B. V,W,X). Als Eingabe-Aufforderung erscheint ein Fragezeichen. Bei der Eingabe können mehrere Werte durch Kommata getrennt werden. Leerräume am Anfang oder Ende werden ignoriert.

INPUT #n,V
Wie INPUT, jedoch erfolgt die Eingabe nicht von der Tastatur, sondern von der Datei mit der Nummer n z.B. von Diskette.

KILL N$
Die Datei N$ auf der Floppy-Disk wird gelöscht. Beispiel: KILL PGM1.BAS.

LET Variable=Wert
Bei Wertzuweisungen ist LET ein optionaler Befehl und wird meist weggelassen.

LINE INPUT, LINE INPUT # n
Wie INPUT bzw. INPUT # n, jedoch kann nur eine einzige Variable eingegangen werden. Dafür sind aber auch Kommata als String-Zeichen erlaubt.

LIST <n><−m>
Auflisten des Programms auf dem Bildschirm. LIST n = nur Zeile n, LIST n−m = Zeilen n bis m, LIST −m = alle Zeilen bis m, LIST n− = alle Zeilen ab n.

LLIST
Wie LIST, die Ausgabe erfolgt aber auf einen angeschlossenen Drucker.

LOAD N$ <,R>
Programm namens N$ von Diskette laden. Mit dem Buchstaben R wird es auch gleich gestartet.

LPRINT
Wie PRINT (siehe unten), jedoch erfolgt die Ausgabe auf den Drucker.

LSET, RSET
Links- und rechtsbündiges Formatieren von Strings. Beispiel:

RSET V$=N$

Diese Befehlsfolge formatiert N$ rechtsbündig in einem Random-Feld V$.

MERGE N$
Mischen zweier Programme: Das Programm N$ wird geladen, ohne ein im Speicher stehendes zu löschen. N$ muß als ASCII-Datei auf der Diskette stehen, also mit SAVE N$,A abgespeichert worden sein.

MID$
Ersetzen eines Teilstrings in einem String. Beispiel:

MID$(A$,4)="AB"

Hier werden das vierte und fünfte Zeichen im String A$ durch die Zeichenfolge AB ersetzt. (Vgl. MID$ als Funktion!)

NAME N1$ AS N2$
Umbenennen einer Disk-Datei. Vorsicht: N1$ und N2$ dürfen bei Basic-80 keine Array-Variablen sein!

NEW
Löschen des Programms und aller Variablen im Speicher.

NULL n
Bei Ausgabeoperationen (LIST, PRINT usw.) werden nach jedem Return-Zeichen n NUL-Steuerzeichen ausgegeben, um z.B. einem Drucker Zeit zum Wagenrücklauf zu geben.

ON ERROR GOTO n
Wenn ein Fehler auftritt, verzweigt das Programm zur Zeile n. Das Programm wird nur abgebrochen, wenn innerhalb der dann ausgeführten Befehlsfolge wieder ein Fehler auftritt.

ON V GOSUB n, m, ... bzw. ON V GOTO n, m, ...
Je nach Wert der Variablen V verzweigt das Programm zu den Zeilen n, m usw. Die Zählung beginnt mit V=1.

OPEN
Öffnen einer Disk-Datei. Je nach Schreib- oder Lese-Option gibt es unterschiedliche Formate:

OPEN "O", <#> n, N$	Öffnen zum Schreiben
OPEN "I", <#> n, N$	Öffnen zum Lesen
OPEN "R", <#> n, N$, L	Random-Datei mit L = Feldlänge

N$ ist hierbei der Filename, n die Dateinummer (mehrere Files können gleichzeitig geöffnet werden!) und L die nur bei Random-Dateien nötige Angabe der Feldlänge. Bei sequentiellen Files entfällt L.

OPTION BASE n
n gibt den minimalen Index von Array-Variablen an, der 0 oder 1 sein kann. Normalerweise ist Null voreingestellt.

OUT n,m
Ausgabe des Bytes m an die Port-Adresse m.

PEEK
Abliefern des Inhalts einer Speicherstelle. V=PEEK(A) liefert den Inhalt der Speicherzelle an der Adresse A in der Variablen V ab.

POKE n,m
Ablegen des Bytes m an der Adresse n. m darf 0...255 sein, n = 0...65535.

PRINT
Ausgabebefehl. Je nach Zahl und Ausgabeformat der nachfolgenden Variablen sind folgende Formate möglich:

PRINT A	A wird mit nachfolgendem Return ausgegeben
PRINT A;	A wird ohne nachfolgendes Return ausgegeben
PRINT A,B	A und B werden tabellarisch ausgegeben
PRINT A;B	A und B werden ohne Abstand ausgegeben
PRINT "TEXT";A	TEXT und A werden ohne Abstand ausgegeben
PRINT "TEXT"A	Gleichbedeutend mit PRINT "TEXT";A

PRINT USING S$;V
Die Variable V wird im Stringfeld S$ kommabündig formatiert. Beispiel:
PRINT USING "# # # . # # ";V
druckt den Wert von V mit drei Vorkomma- und zwei Nachkommastellen.
PRINT USING "# # # # # ,. # # ";V
druckt V mit zwei Nachkommastellen und trennt die Tausenderstelle mit einem Komma, z.B. 1,236.56.
PRINT USING "# # . # # ^^^^ ";V
druckt V im Exponentialformat, z.B. als 1.23E+04.

PRINT # n
Wie PRINT, die Ausgabe erfolgt jedoch nicht auf den Bildschirm, sondern auf die Datei mit der Nummer n. Auch das Format PRINT # n USING ist möglich.

PUT <#> n <,m>
Die Disk-Puffer werden auf die Random-Datei mit der Nummer n und der Record-Nummer m geschrieben.

RANDOMIZE <n>
Der Zufallszahlen-Generator (RND) wird zurückgesetzt und mit der Zahl n initialisiert. Fehlt n, so fragt der Interpreter den Benutzer automatisch nach einer Anfangszahl.

READ V
Aus einer DATA-Anweisung wird ein Wert gelesen. Statt V dürfen auch mehrere Variable hinter READ stehen, durch Kommata voneinander getrennt.

REM
Ein nach REM stehender Text wird bis zum Zeilenende ignoriert. REM sollte nicht hinter DATA stehen, weil der Befehl dann als String aufgefaßt würde.

RENUM <a> <,b,c>
Neunumerieren des Basic-Programms im Speicher mit a = neue Anfangs-Zeilennummer, b = erste bisherige umzunumerierende Zeile und c = Schrittweite (Inkrement).

RESET
Initialisiert die Disk-Laufwerke, z.B. nach Diskettenwechsel unter CP/M 2.2.

RESTORE <n>
Zurücksetzen des DATA-Zählzeigers auf Null oder optional auf eine bestimmte Zeilennummer n, um die DATA-Werte nochmals mit READ lesen zu können.

RESUME
Wiederaufnahme der Programmausführung nach einem ON-ERROR-Befehl, d.h. nach einer Fehlermeldung. Mögliche Formate sind:

RESUME oder RESUME 0	Das Programm wird dort fortgesetzt, wo der Fehler auftrat.
RESUME NEXT	Das Programm wird beim nächsten Befehl fortgesetzt.
RESUME n	Das Programm wird bei einer bestimmten Zeilennummer fortgesetzt.

RUN <n> oder RUN N$
Alle Variablen werden gelöscht und das Programm gestartet. Optional kann eine Zeilennummer angegeben werden, an der die Ausführung beginnen soll. (Alternative ohne Löschen von Variablen: GOTO n!) RUN N$ lädt das Programm N$ von Diskette und startet es.

SAVE N$ <,A> <,P>
Abspeichern des Programms im Speicher auf Diskette. Durch Anhängen von A wird es nicht in komprimierter Token-Form, sondern als ASCII-Datei abgespeichert, mit P als LIST-geschützte Datei. An den Namen N wird die CP/M-Extension .BAS angefügt.

STOP
STOP wirkt ähnlich wie END, führt aber zu der Meldung BREAK IN n mit n=Zeilennummer, wie sie auch bei einem Abbruch mit CTRL-C auftritt. Die Programmausführung kann mit CONT wieder aufgenommen werden.

SWAP X,Y
Die beiden Variablen X und Y werden miteinander vertauscht. Beide müssen natürlich vom gleichen Typ sein.

TRON, TROFF
Nach TRON wird während der Programmausführung die gerade bearbeitete Zeilennummer auf den Bildschirm ausgegeben (Trace-Modus). Dieser Modus kann mit TROFF wieder beendet werden.

WAIT x,y <,z>
Das Programm wartet, bis am Port mit der Adresse x die Daten gleich y sind. Wenn z angegeben ist, wird y vorher mit z exklusiv-oder-verknüpft.

WHILE...WEND
Diese Anweisungen erlauben sauber strukturierte Programmschleifen und sind als Alternative zu IF...THEN...GOTO zu verstehen. Ein Beispiel:

```
WHILE A=B: PRINT "A IST GLEICH B": WEND
```

WIDTH n
Nach der Ausgabe von n Zeichen in einer Zeile generiert der Interpreter ein Return-Zeichen. Die Anweisung WIDTH LPRINT n wirkt ausschließlich auf den Druckertreiber.

WRITE A$ und WRITE # n,A$
Ähnlich PRINT A$ und PRINT # n,A$, Ausgabe von Strings in Anführungszeichen.

Funktionen

ABS
Absolutwert eines Ausdrucks bilden, d.h. eventuell negatives Vorzeichen durch positives ersetzen. Beispiel:

```
A=ABS(X)
```

ASC
Zeichencode eines Strings liefern; wenn der String länger als ein Zeichen ist, wird nur das erste Zeichen berücksichtigt. Beispiel:

```
A=ASC(X$)
```

ATN
Arcustangens im Bogenmaß berechnen. Beispiel:

```
A=ATN(X)
```

CDBL
Umwandlung einer Zahl einfacher Genauigkeit in eine doppelt genaue. Beispiel:

```
DEFDBL B:B=CDBL(A)
```

CHR$
Zeichen als String aus Code generieren, z.B. um spezielle Steuerzeichen auszugeben, die über die Tastatur nicht erreichbar sind. Beispiel:

```
PRINT CHR$(10);
```

CINT
Umwandlung einer Fließkomma- in eine Ganzzahl-Variable, wobei der Nachkommateil gerundet wird. Beispiel:

```
A%=CINT(B)
```

COS
Cosinus des im Bogenmaß-Arguments errechnen (nur mit einfacher, d.h. sechsstelliger Genauigkeit möglich). Beispiel:

```
A=COS(B)
```

CSNG
Umwandlung einer Integer-Variable oder doppelt genauen Variable in eine mit einfacher Genauigkeit. Beispiel:

```
B=CSNG(A#)
```

CVI, CVS, CVD
Umwandlung eines 2-, 4- bzw. 8-Zeichen-Strings in eine numerische Variable, z.B. zum Lesen einer Random-Datei mit numerischen Werten von Diskette. Beispiel:

```
A=CVS(X$)
```

EOF
End-of-File-Kontrollfunktion. Der Befehl nach IF EOF(n) wird ausgeführt, wenn die mit der Nummer n geöffnete Datei bis zum Ende gelesen wurde. Beispiel:

```
90 OPEN "I",1,"NAME"
100 IF NOT EOF(1) THEN INPUT #1,A$:PRINT A$:GOTO 100
110 PRINT "FERTIG":CLOSE 1
```

EXP
Errechnen der e-Funktion, wobei das Argument maximal 87.3365 sein darf, da sonst ein Überlauf auftritt. Beispiel:

```
PRINT EXP(3)
```

FIX
Liefert den links vom Komma stehenden, ganzzahligen Teil einer Zahl. Der Unterschied zu INT ist, daß FIX bei negativen Argumenten nicht die nächst niedrigere Zahl liefert. Beispiel:

PRINT FIX(−58.43) ergibt −58 (bei INT:−59!)

FRE(0), FRE("")
Feststellen des freien Speicherplatzes. FRE("") bzw. FRE(X$) rechnet hierbei den von nicht mehr benötigten Strings („Garbage") benötigten Speicherplatz nicht mit, indem eine „Garbage Collection" (Müllbeseitigung) durchgeführt wird, die einige Sekunden dauern kann. Die hinter FRE angegebene Variable ist nicht relevant (Dummy). Beispiel:

```
PRINT FRE(X$)
```

HEX$
Liefert das hexadezimale Äquivalent des Arguments als String. Beispiel:
PRINT HEX$(65535) ergibt FFFF.

INKEY$
Liefert einen 1-Zeichen-String mit dem Zeichen, das gerade auf der Tastatur gedrückt wurde, oder einen leeren String (Länge 0), wenn keine Taste gedrückt wurde. Beispiel:

```
100 A$=INKEY$:IF INKEY$="" THEN 100
110 PRINT"GEDRUECKTE TASTE =";A$
```

INP
Liefert ein Byte von einer Portadresse (vgl. OUT-Befehl!). Beispiel:
A=INP(51)

INPUT$
Liefert einen Zeichenstring entweder von der Tastatur oder von einer Datei, der nicht mit Return beendet wird, sondern durch Erreichen einer vorher spezifizierten Anzahl von Zeichen. Beispiele:

A$=INPUT$(n)	liest ein Zeichen von der Tastatur
A$=INPUT$(n,m)	liest n Zeichen von der Datei Nr. m

INSTR
Liefert die Position des ersten Auftretens eines Strings in einem anderen String oder Null, falls er nicht im anderen vorkommt. Optional kann eine Position angegeben werden, ab der gesucht werden soll. Beispiel:

PRINT INSTR("ABCDEB","B")	ergibt 2
PRINT INSTR(4,"ABCDEB","B")	ergibt 6

LOF(n)
Liefert die Länge der Datei n im Puffer. Beispiel:
IF LOF(1)>0 THEN INPUT # 1,A$

LOG
Liefert den natürlichen Logarithmus einer Zahl, die größer als Null sein muß. Beispiel:
PRINT LOG(3)

LPOS
Liefert die momentane Position des Druckkopfes eines angeschlossenen Drukkers. Bei Arbeiten mit Drucker-Pufferspeichern entspricht dies allerdings nicht unbedingt der tatsächlichen, physikalischen Position. X ist eine Dummy-Variable, die nur für die Syntax nötig ist. Beispiel:
PRINT LPOS(X)

MID$
Bildet einen Teilstring aus einem String. Beispiele für das Format:

A$=MID$(B$,2)	liefert in A$ den String B$ ab dem zweiten Zeichen, d.h. das erste wird abgeschnitten.
A$=MID$(B$,2,3)	liefert in A$ die drei Zeichen, die in B$ ab der zweiten Position stehen.

MKI$, MKS$, MKD$
Aus einem numerischen Argument wird ein String aus 2, 4 oder 8 Zeichen gewonnen. Beispiel:

```
A$=MKS$(B)
```

OCT$
Wie HEX$, es wird jedoch nicht ein Hexadezimal-, sondern ein Oktalstring erzeugt.

PEEK
Liefert den Inhalt einer Speicherzelle (0...255). Beispiel:

```
PRINT PEEK(65530)
```

POS
Liefert die momentane Cursor-Position (ganz links entspricht 1). X ist eine Dummy-Variable, jeder Name ist geeignet. Beispiel:

```
Y=POS(X)
```

RIGHT$
Liefert den rechten Teil eines Strings. Beispiel:

A$=RIGHT$(B$,3)	liefert die rechten drei Zeichen von B$.

RND
Zufallsfunktion; die Initialisierung kann mit RANDOMIZE erfolgen. Folgende Möglichkeiten gibt es:

A=RND	Liefert jedesmal eine neue Zufallszahl.
A=RND(4)	Mit beliebigen positiven Zahlen als Argument wirkt RND genauso wie ohne Argument.
A=RND(0)	Wiederholt die letzte Zufallszahl.
A=RND(−5)	Mit negativen Argumenten wird die Zufallszahlenfolge wieder von vorn begonnen.

Natürlich gibt es bei Computern keine echten Zufallszahlen. Vielmehr wird eine Pseudo-Zufallszahlen-Folge durch Verschieben eines Bitmusters in einigen Speicherzellen gewonnen.

SGN
Liefert bei negativen Argumenten −1, bei positiven +1 und bei Null den Wert 0. Beispiel:

```
PRINT SGN(A)
```

SIN
Liefert den Sinus eines im Bogenmaß angegebenen Winkels mit einfacher Genauigkeit. Beispiel:

```
PRINT SIN(3.1415)
```

SPACE$
Liefert einen String mit der im Argument angegebenen Zahl von Leerräumen. Beispiel:

```
A$=SPACE$(20)+"HIER":PRINT A$
```

SPC
Gibt eine Anzahl von Leerräumen aus. Beispiel:

```
PRINT SPC(20);"HIER"
```

SQR
Liefert die Quadratwurzel eines positiven Arguments. Beispiel:

```
A=SQR(B)
```

STR$
Umwandlung einer numerischen in eine String-Variable (vgl. VAL). Beispiel:

```
A$=STR$(B)
```

STRING$
Liefert eine bestimmte Anzahl des spezifizierten Zeichens. Wenn der String im Argument länger als ein Zeichen ist, wird nur das erste berücksichtigt. Beispiel:

PRINT STRING$(5,"A") oder PRINT STRING$(5,65) liefert fünf A.

TAB
Setzen des Cursors auf eine bestimmte Position in der Zeile. Wenn diese links von der momentanen Cursor-Position ist, geht TAB automatisch in eine neue Zeile. Beispiel:

```
PRINT"WERT";TAB(20);A;"DM"
```

TAN
Liefert den Tangens eines im Bogenmaß angegebenen Winkels. Wenn das Ergebnis größer als die größte darstellbare Zahl ist, wird „Overflow Error" ausgegeben und das Ergebnis gleich der maximal darstellbaren Zahl gesetzt; die Programmausführung wird hierbei nicht beendet. Beispiel:

```
A=TAN(3.1415/4)
```

USR
Aufruf eines Maschinenprogramms, dessen Startadresse vorher mit DEFUSR festgelegt werden muß. Optional können nach USR eine Programm-Nummer sowie ein oder mehrere zu übergebende Parameter stehen, andernfalls ist eine Dummy-Variable einzusetzen. Beispiele:

```
A=USR(X)      Aufruf mit Dummy-Variable.
A=USR 2(C,D)  Programm 2 wird mit C und D als Parameter aufgerufen.
```

VAL
Umwandeln einer String- in eine numerische Variable. Beispiel:
A=VAL(B$)

VARPTR
Liefert die Adresse, an der eine bestimmte Variable gespeichert ist. Bei String-Variablen wird nicht die Adresse des Strings selbst, sondern die des Zeigers auf ihn geliefert. Beispiel:
A=VARPTR(B$)

VARPTR#(n)
Liefert die Startadresse des Disk-Puffers, der zur Datei mit der Nummer n gehört.

Operatoren

Operator	Zweck	Beispiele
=	Gleichheit	A=B:IF A=10 THEN PRINT "B=10"
<	Kleiner	IF A<0 THEN PRINT "A ist negativ"
>	Größer	IF A>0 THEN PRINT "A ist positiv"
<>	Ungleich	IF A<>B THEN PRINT "A ist nicht B"
<=	Kleiner/gleich	IF A<=5 THEN PRINT "A=0 oder neg."
>=	Größer/gleich	IF A>=0 THEN PRINT "A=0 oder pos."
+	Addition	PRINT 10+3
+	String-Add.	A$=A$+"?"
−	Subtraktion	PRINT 10−3
	oder Negation	A=5:A=−A:PRINT A
*	Multiplikation	PRINT 10*3
/	Division	PRINT 10/3
↑	Exponentiation	PRINT 22↑16
NOT	Log. Negation	IF NOT EOF(1) THEN ...
AND	Bitweises Und	A=IN(5) AND 3:OUT 5,A
OR	Bitweises Oder	IF A=0 OR B=0 THEN ...
XOR	Exklusiv-Oder	OUT 5,IN(5) XOR 255
IMP	Implizierung	C=A IMP B
EQV	Äquivalenz	IF A EQV B THEN PRINT "Alle Bits gleich"

Hinweise: Das Multiplikationszeichen darf zwar in der Algebra, nicht aber in Basic weggelassen werden. Ein Ausdruck wie y=4x muß in Basic also als Y=4*X geschrieben werden. Die algebraische Hierarchie (Punkt vor Strich; z.B. 5+2*3↑2=23) wird berücksichtigt und kann mit Klammern umgangen werden. Die bitweisen Verknüpfungen arbeiten nur mit 16-Bit-Werten. Vergleichs-Operatoren sind auch auf Strings anwendbar; es gilt die Reihenfolge des ASCII-Alphabets, Kleinbuchstaben sind also „größer" als Großbuchstaben, und „ä" ist größer als „z".

Variablendarstellung

A%	Integer-Variable (Ganzzahl-Variable, −32768...+32767)
A oder A!	Variable mit einfacher Genauigkeit: maximal 7stellige Mantisse und zweistelliger Exponent
A#	Variable mit doppelter Genauigkeit (max. 16 Stellen)
A$	Stringvariable, 0...255 Zeichen
A(I)	Eindimensionale Array-Variable
A(I,J)	Zweidimensionale Array-Variable

Alle Variablennamen müssen mit einem Buchstaben beginnen. Bei einfachen Interpretern (8-KByte-Version) sind nur die ersten beiden Zeichen (außer #,$,%,!) relevant, d.h. die Namen ABC und ABD rufen die gleiche Variable auf. Die „größeren" MBasic-Versionen lassen 40 signifikante Zeichen zu. Variablennamen dürfen keine reservierten Worte (d.h. Basic-Befehle) sein. FN ist für den Aufruf einer Funktion reserviert, die mit DEF FN definiert wurde. Die Typen-Definition kann statt mit den Zeichen %, !, # und $ auch am Programmanfang durch DEFINT, DEFSNG, DEFDBL und DEFSTR erfolgen.

Schreibweise von Zahlen

123	Dezimalzahl 123
1.23	Dezimalzahl 1,23 (Dezimalpunkt statt Komma!)
21E6	Dezimalzahl 21*10↑6 = 21 000 000 = 2.1*10↑7
5E−3	Dezimalzahl 5*10↑−3 = 0.005
&H76	Hexadezimalzahl 76
&54, &O54	Oktalzahl 54 (O wie Otto!)

Erlaubter Wertebereich bei Fließkomma-Berechnungen: 1E−38...1E+38. Bei einfacher Genauigkeit werden intern sieben Stellen berechnet, jedoch nur sechs ausgegeben.

Applesoft-Basic

Der von Apple in Cupertino gebaute Apple-II besaß zunächst einen Ganzzahl-Interpreter, später wurde er jedoch mit dem sogenannten Applesoft-Interpreter geliefert. Dieser Interpreter für die CPU 6502 wurde wie MBasic von Microsoft entwickelt und enthält einerseits einen Teil der MBasic-Befehle, andererseits aber auch einige zusätzliche, u.a. für das Erstellen von Grafiken.

Die folgenden Befehle sind auch in MBasic vorhanden und werden hier daher nicht neu erläutert:

ABS	ASC	ATN	CALL	CHR$
CLEAR	CONT	COS	DATA	DEF FN
DIM	END	EXP	FOR/TO/STEP/NEXT	
FRE(0)	GOSUB	GOTO	IF/THEN	INPUT
INT	LEFT$	LEN	LET	LIST
LOAD	LOG	MID$	NEW	PEEK
POKE	PRINT	READ	REM	RESTORE
RESUME	RETURN	RIGHT$	RND	RUN
SAVE	SGN	SIN	SPC	SQR
STOP	STR$	TAB	TAN	USR
VAL	WAIT			

Allerdings gibt es bei der Syntax kleine Abweichungen. Nach CLEAR kann beim Apple-II kein Argument stehen, so daß man damit z.B. nicht die RAM-Obergrenze für Basic setzen kann (dafür gibt es HIMEM, siehe unten). Nach RESTORE kann man keine Zeilennummer angeben, das nächste READ-Kommando fängt also bei der allerersten DATA-Anweisung neu an. Zu USR gibt es keine Adressen-Zuweisung wie DEFUSR, vielmehr muß man den zugehörigen Sprungbefehl zum Maschinen-Unterprogramm (JMP) ab der Adresse 000A eintragen. Nach RESUME darf kein NEXT stehen, d.h. das Programm wiederholt nach RESUME die Anweisung, bei der ein Fehler auftrat. INPUT gibt kein Fragezeichen aus, wenn ein Text in Anführungszeichen steht. Die speziellen Applesoft-Befehle seien nun ebenfalls kurz erläutert.

COLOR = n
Setzt die Farbe für Grafiken niederer Auflösung. Folgende Farben sind möglich:

0	schwarz	4	dunkelgrün	8	braun	12	grün
1	magenta	5	grau	9	orange	13	gelb
2	dunkelblau	6	blau	10	grau	14	blaugrün
3	purpur	7	hellblau	11	violett	15	weiß

Der GR-Befehl setzt die Farbe auf Null.

DEL
Dieser Befehl entspricht dem MBasic-Befehl DELETE (Löschen von Zeilen im Programm).

DRAW n <AT x,y>
Zeichnen des hochauflösenden Grafik-Shapes n an der Position x, y oder, falls diese nicht angegeben ist, ab der Position, an der zuletzt gezeichnet wurde.

FLASH
Nach dem FLASH-Befehl werden alle PRINT-Ausgaben blinkend auf dem Bildschirm dargestellt (nicht bei Kleinbuchstaben möglich).

GET
Dieser Befehl wartet, bis eine Taste gedrückt wird, und liefert ihren numerischen Wert in einer numerischen bzw. das ASCII-Zeichen in einer String-Variablen. Beispiele:

```
GET A$:PRINT A$
GET A:PRINT A
```

Die Wirkung ähnelt also dem INKEY$-Befehl in MBasic, allerdings ist die Syntax völlig anders, und INKEY wartet auch nicht auf einen Tastendruck (ähnlich wie GET bei Commodore-Basic).

GR
Der Bildschirm wird von Textdarstellung auf niedrig auflösende Grafik umgeschaltet (40 × 40 Punkte). Am unteren Rand bleiben vier Textzeilen zur Schriftausgabe.

HCOLOR = n
Angabe der Farbe für das Zeichnen hochauflösender Grafiken. Die folgenden Werte sind dafür möglich:

0	schwarz	4	schwarz
1	grün	5	violett
2	blau	6	rot
3	weiß	7	weiß

Bedingt durch die sehr einfache Erzeugung der Farben im Apple-Video-Interface ist diese Tabelle jedoch nur als Anhaltspunkt zu betrachten, da die tatsächlichen Farbwerte davon abweichen können.

HGR, HGR2
Umschalten auf hochauflösende Grafik. HGR stellt den Speicherbereich hex 2000...3FFF als Grafikseite dar und läßt am unteren Bildschirmrand vier Textzeilen stehen. HGR2 stellt den Bereich 4000...5FFF als Grafik-Seite 2 dar und füllt damit den gesamten Bildschirm.

HIMEM:
Setzen der für Basic-Programme und -Variablen zur Verfügung stehenden RAM-Obergrenze, z.B. HIMEM:38400.

HLIN x1,x2 AT y
Zeichnen einer horizontalen Linie in niedrig auflösender Grafik von x1,y nach x2,y.

HOME
Löschen des Textbildschirms. Der Cursor wird dabei in die linke obere Ecke gesetzt.

HPLOT x,y
HPLOT TO x,y
HPLOT x1,y1 TO x2,y2 <TO x3,y3...>
Setzen eines Punktes (HPLOT x,y) oder Zeichnen einer Linie zwischen zwei Punkten in hochauflösender Grafik. Durch Wiederholen von TO können in einem einzigen Befehl ganze Grafiken erstellt werden, ein den Bildschirm umschließendes Rechteck z.B. mit dem Befehl

HPLOT 0,0 TO 279,0 TO 279,159 TO 0,159 TO 0,0

HTAB n
Der Cursor wird in die n-te Bildschirmspalte gesetzt, wobei die Zählung für n ganz links mit 1 beginnt.

INVERSE
Nach dem INVERSE-Befehl erfolgen alle PRINT-Ausgaben in inverser Schrift, also schwarz in einem weißen Feld. Vorsicht: Beim Apple-II lassen sich Kleinbuchstaben so nicht invers oder blinkend darstellen!

IN# n
Die Tastatur-Eingabe wird auf die in Slot n steckende I/O-Karte umgeschaltet.

LOMEM:
Die Speicher-Untergrenze für die Variablen wird gesetzt; normalerweise ist sie gleich dem Ende des Basic-Programms. Beispiel:

LOMEM:16384

setzt die RAM-Untergrenze oberhalb der HGR-Seite 1, so daß die Variablen nicht Grafiken überschreiben können.

NORMAL
Rücksetzen der FLASH- oder INVERSE-Darstellung. Nur im Normalmodus können auch Kleinbuchstaben dargestellt werden, sofern der eingebaute Zeichengenerator dies erlaubt (bei älteren Apples ist das nicht der Fall).

NOTRACE
Der TRACE-Modus wird wieder ausgeschaltet, d.h. das Programm wird nach RUN normal interpretiert. NOTRACE darf auch innerhalb des Programms stehen.

ONERR GOTO
Sprungbefehl, der beim Auftreten von Fehlern ausgeführt wird. Die Art des Fehlers steht codiert in der Speicherzelle dezimal 222, kann also mit PEEK(222) ermittelt werden:

0	NEXT ohne FOR	69	Zu große Zahl
16	Syntax-Fehler	77	Speicher voll
22	RETURN ohne GOSUB	90	Zeilennummer falsch
42	Keine DATAs mehr	107	Array-Index zu groß
53	Unzulässiger Wert	120	Array zweimal dimensioniert

133	Division durch Null	224	Funktionsaufruf ohne DEF FN
163	Variablentyp paßt nicht	254	Unzulässige Eingabe bei INPUT
176	String mit mehr als 255 Zeichen	255	Abbruch mit CTRL-C versucht
191	zu komplexer Ausdruck		

Nach ONERR wird der Stackpointer zurückgesetzt. Das Programm darf deshalb nicht innerhalb eines Unterprogramms oder in einer FOR-NEXT-Schleife fortgesetzt werden, d. h. RESUME ist hier problematisch.

PDL n
Liefert einen Wert von 0...255, der die Stellung des Joysticks mit der Nummer n (0...3) angibt. Zwei unterschiedliche Joysticks sollten nicht unmittelbar nacheinander, sondern mit einer kurzen Pause gelesen werden (einige zehn ms), da sonst falsche Ergebnisse auftreten.

PLOT x,y
Zeichnen eines Punktes in niedrig auflösender Grafik.

POP
Rücksetzen des Stackpointers um einen GOSUB-Befehl. Nach POP ist es möglich, unmittelbar aus einem Unterprogramm mit GOTO statt RETURN in das Hauptprogramm zurückzuspringen.

POS(0)
Liefert die horizontale Position des Cursors in einer Bildschirmzeile. 0 ist hier ein Dummy-Wert ohne weitere Bedeutung. Beispiel:

```
PRINT SPC(20);POS(0)
```

PR# n
Umleiten der Bildschirm-Ausgabe auf die I/O-Karte in Slot n oder Initialisieren einer Karte. Beispiel: PR# 6 lädt das Disk-Betriebssystem (Boot-Vorgang), wenn in Slot 6 die Floppy-Controller-Karte steckt. PR# 1 leitet alle Ausgaben auf eine in Slot 1 steckende Drucker-Interface-Karte um.

RECALL A$
Lädt das Feld A$() von Kassette (bei Floppy-Disk nicht anwendbar).

ROT = n
Angabe der Drehung, mit der beim nächsten DRAW-Befehl ein Grafik-Shape gezeichnet werden soll. 360 Grad entspricht ROT=64. ROT=16 entspricht einer Drehung von 90 Grad im Uhrzeigersinn.

SCALE = n
Angabe des Maßstabs, mit der beim nächsten DRAW-Befehl ein Grafik-Shape gezeichnet werden soll. n kann 0...255 betragen (1 ist Normalgröße, 2 ist doppelt so groß usw.).

SCRN (x,y)
Liefert den Farbcode des Punktes an der Stelle x, y bei niedrig auflösender Grafik.

SHLOAD
Laden einer Grafik-Shape-Definition von Kassette (bei Floppy-Betrieb nicht möglich, Ersatz durch BLOAD = Laden eines Speicherbereichs).

SPEED n
Ändern der Ausgabe-Geschwindigkeit von LIST und PRINT. 255 ist der Maximalwert, 0 führt zu extrem langsamer Ausgabe. Für bequemes Mitlesen ist SPEED=200 ein günstiger Wert.

STORE A$
Abspeichern des Arrays A$() auf Kassette (bei Floppy-Betrieb nicht möglich).

TEXT
Rückschalten vom GR- oder HGR-Modus auf reine Textdarstellung mit 40 Zeichen pro Zeile und 24 Zeilen.

TRACE
Wie TRON in MBasic, Ausgabe der Zeilennummern während der Programmausführung.

VLIN y1,y2 AT x
Zeichnen einer vertikalen Linie von x,y1 nach x,y2 bei niedrig auflösender Grafik.

VTAB n
Der Cursor wird in die n-te Zeile des Bildschirms gesetzt; die Zählung beginnt ganz oben mit 1. Die horizontale Position wird dabei nicht verändert.

XDRAW n AT x,y
Wie DRAW, allerdings wird nicht die mit HCOLOR spezifizierte Farbe zum Zeichnen des Shapes verwendet, sondern die „übermalten" Punkte werden mit dem neuen Shape Exklusiv-Oder-verknüpft, d.h. umgedreht. Damit ist es auch möglich, ein Shape wieder zu löschen, indem man es nach DRAW an der gleichen Position mit XDRAW invertiert.

Shapes

Bei den Befehlen DRAW, ROT, SCALE und XDRAW wurden sogenannte Shapes erwähnt. Das sind Grafik-Figuren, die als Bitmuster irgendwo im RAM abgelegt sind und von speziellen Routinen innerhalb des Basic-Interpreters in unterschiedlichen Winkeln, Farben und Maßstäben in die jeweils aktivierte RAM-Grafik-Seite und damit auf den Bildschirm transferiert werden können. (Demgegenüber erfolgt die Darstellung von Sprites z.B. beim C-64 nicht durch Software, sondern mit Hilfe spezieller ICs im Video-Interface, also per Hardware).

Beim Apple-II kann man bis zu 255 Shapes gleichzeitig im Speicher verwalten. Die dazu nötige „Shape-Tabelle" ist wie folgt aufgebaut (als Basisadresse wird hier hex 8000 angenommen):

Adresse	Inhalt
8000	Gesamtzahl der Shapes
8001	nicht verwendet
8002	Zeiger auf erstes Shape, niederw. Byte
8003	Zeiger auf erstes Shape, höherw. Byte
8004	Zeiger auf zweites Shape, niederw. Byte
8005	Zeiger auf zweites Shape, höherw. Byte
usw.	usw.
8XXX	Bitmuster für Shape 1, letztes Byte = 00
8YYY	Bitmuster für Shape 2, letztes Byte = 00
usw.	usw.

Das Bitmuster eines einzelnen Shapes besteht aus mehreren Bytes, wovon das letzte immer Null sein muß. Jedes Byte ist folgendermaßen aufgebaut, wobei DRAW und XDRAW die acht Bits von rechts nach links abarbeiten (das niederwertigste also zuerst):

Bit	7	6	5	4	3	2	1	0
Bedeutung	R	R	S	R	R	S	R	R

Ein Paar von R-Bits gibt eine von vier möglichen Richtungen an, während die mit S gekennzeichneten Bits angeben, ob der dann erreichte Punkt sichtbar oder unsichtbar sein soll:

RR	Richtung	S	Bedeutung
00	nach oben	0	Unsichtbarer Punkt
01	nach rechts	1	Sichtbarer Punkt
10	nach unten		
11	nach links		

Dem ganz links stehenden RR-Bitpaar fehlt ein S-Bit, da ein Byte ja nur acht Bits besitzt. Das zugehörige S-Bit wird hier als Null angenommen, der zuletzt durch dieses Steuerbyte angefahrene Punkt bleibt also unsichtbar. Ferner muß man darauf achten, daß ein Byte, das 00 ergibt, ignoriert wird, weil es als Endmarkierung eines Shapes dient. Dieser Fall träte auf, wenn ein Byte aus lauter als unsichtbar gekennzeichneten Aufwärts-Bewegungen bestünde, und muß daher vermieden werden. Sobald die gesamte Shape-Tabelle fertiggestellt ist, muß ihre Anfangsadresse (in obigem Beispiel hex 8000) in die Zero-Page-Adressen E8 (niederwertiger Teil, hier 00) und E9 (höherwertiger Teil, hier hex 80 = dezimal 128) geschrieben werden. Das kann auch vom Basic-Programm aus geschehen, hier mit den Befehlen POKE 232,0:POKE 233,128.

Beim Apple-II ist es nicht ohne weiteres möglich, eine auf dem Bildschirm stehende Grafik zu beschriften, da höchstens am unteren Bildschirmrand noch vier Textzeilen zur Verfügung stehen, nicht aber auf der übrigen Grafik-Fläche. Einen Ausweg stellt die Verwendung einer Shape-Tabelle dar, die für jedes ASCII-Zeichen ein entsprechendes Shape definiert. *Abb. 3.2.3.1* zeigt eine solche Shape-Ta-

```
7000- 60 00 C2 00 C5 00 CD 00 D5 00 E7 00 FA 00 0A 01 +=0575
7010- 1A 01 20 01 2A 01 34 01 48 01 54 01 5D 01 65 01 +=01FE
7020- 6D 01 78 01 8B 01 96 01 A4 01 B2 01 C1 01 D1 01 +=04F6
7030- E3 01 FE 01 0F 02 21 02 28 02 31 02 3E 02 48 02 +=02FE
7040- 53 02 5E 02 6E 02 7E 02 8E 02 9D 02 AB 02 BB 02 +=043E
7050- C9 02 D8 02 E7 02 F2 02 FE 02 11 03 1A 03 2C 03 +=04E2
7060- 3D 03 4B 03 58 03 69 03 7B 03 8C 03 96 03 A2 03 +=03A0
7070- AE 03 BD 03 CF 03 DC 03 E9 03 FB 03 04 04 14 04 +=052C
7080- 1E 04 28 04 2C 04 3A 04 49 04 58 04 68 04 79 04 +=024E
7090- 87 04 97 04 A4 04 B0 04 BD 04 CB 04 D4 04 E2 04 +=05D0
70A0- F0 04 FD 04 0C 05 1B 05 27 05 36 05 44 05 50 05 +=032B
70B0- 5B 05 7A 05 89 05 97 05 A4 05 B0 05 B8 05 C3 05 +=04EC
70C0- CC 05 01 00 00 09 09 36 36 16 06 00 00 09 31 2E +=01D4
70D0- 21 2C 00 00 00 09 31 36 36 2E 21 24 24 2C 12 3F +=0207
70E0- 3F 16 2D 2D 06 00 00 09 09 0D 3A 3F 17 0E 2D 15 +=01B4
70F0- 1E 3F 37 09 24 24 24 05 00 00 09 35 37 12 22 21 +=01D8
7100- 21 21 21 32 12 3A 2E 06 00 00 09 32 16 36 29 0D +=01D2
7110- 1C 0C 1F 1C 0C 24 3B 00 00 00 09 09 36 06 00 00 +=011C
7120- 09 09 1E 1E 36 0E 0E 05 00 00 09 09 15 15 36 1E +=0135
7130- 1E 07 00 00 09 09 36 36 36 3C 23 21 21 25 39 3B +=0213
7140- 33 29 2E 2A 32 00 00 00 09 09 32 36 26 3C 2F 2D +=021E
7150- 35 00 00 00 09 09 12 12 36 3B 00 00 00 09 12 2A +=0121
7160- 2D 35 00 00 00 09 09 12 12 12 06 00 00 09 09 09 +=00CB
7170- 3A 3A 3A 3A 3A 00 00 00 09 32 36 36 29 25 21 24 +=025C
7180- 24 3B 2F 2D 32 37 33 3B 00 00 00 09 11 25 36 36 +=023D
7190- 36 3D 3F 00 00 00 09 22 29 2D 32 1E 3F 3A 32 2D +=025B
71A0- 2D 06 00 00 09 2D 2D 36 3B 2A 15 36 3B 27 3B 00 +=0219
71B0- 00 00 09 09 31 36 36 26 2C 3F 3F 24 21 21 00 00 +=01E5
71C0- 00 09 2D 2D 3F 3F 36 2D 2D 32 36 3B 27 3B 00 00 +=0276
71D0- 00 09 09 2D 3F 17 17 36 2E 11 2F 2D 23 21 1C 3F +=021C
71E0- 07 00 00 09 2D 2D 36 33 33 33 36 00 00 00 09 29 +=01A1
71F0- 2D 32 16 36 3B 27 23 04 34 0E 2D 05 00 00 09 29 +=01DA
7200- 2D 32 16 36 3B 27 23 0C 1C 34 15 2D 05 00 00 09 +=01DC
7210- 29 2D 32 3E 3F 1C 24 32 0D 09 36 1E 1E 3F 07 00 +=0245
7220- 00 09 09 12 16 06 00 00 09 11 11 16 36 3B 00 00 +=00F2
7230- 00 09 09 39 31 33 33 33 31 31 29 00 00 00 09 12 +=01BB
7240- 2D 2D 16 3F 3F 07 00 00 09 31 29 32 31 33 33 3B +=025C
7250- 00 00 00 09 22 29 2D 3A 3A 32 16 06 00 00 09 29 +=0175
7260- 2D 32 3E 3E 24 1C 33 36 36 29 2D 04 00 00 09 09 +=0226
7270- 15 15 36 36 24 3F 3F 36 24 24 0C 04 00 00 09 2D +=01FC
7280- 2D 32 16 36 3B 3F 24 24 34 2E 2D 05 00 00 09 29 +=0233
7290- 2D 22 3B 3F 32 36 36 29 25 21 00 00 00 09 2D 2D +=0239
72A0- 32 36 36 3B 3F 24 24 24 00 00 00 09 2D 2D 3F 3F +=0265
72B0- 36 2E 2D 3F 37 36 2D 2D 04 00 00 09 2D 2D 3F 3F +=027C
72C0- 36 2E 2D 3F 37 36 06 00 00 09 29 2D 3F 3F 32 36 +=0288
72D0- 36 29 2D 24 3F 00 00 00 09 36 36 36 24 2C 2D 25 +=023C
72E0- 24 36 36 36 07 00 00 09 29 3D 36 36 36 3D 3F 00 +=025A
72F0- 00 00 09 09 09 36 36 36 3B 27 3B 00 00 00 09 36 +=0199
```

Abb. 3.2.3.1 Shape-Tabelle zur Darstellung von ASCII-Zeichen in hochauflösender Grafik beim Apple-II

belle. Ein kurzes Basic-Programm, das zu Testzwecken alle ASCII-Zeichen als Shapes ausgibt, zeigt *Abb. 3.2.3.2.*

```
7300- 36 36 1D 24 2C 0C 0C 0C 1F 13 12 15 15 15 07 00 +=0187
7310- 00 09 36 36 36 2D 2D 04 00 00 09 36 36 36 24 24 +=01FC
7320- 24 15 15 26 0C 0C 36 36 36 07 00 00 09 36 36 36 +=01E0
7330- 24 24 35 31 29 24 24 36 36 36 07 00 00 09 32 36 +=0239
7340- 36 29 25 21 24 24 3B 3F 00 00 00 09 2D 2D 32 1E +=021A
7350- 3F 27 34 36 36 06 00 00 09 29 2D 32 36 16 1C 1C +=0221
7360- 16 3F 21 23 24 24 00 00 00 09 2D 2D 32 1E 3F 27 +=01FA
7370- 34 36 36 0D 09 1C 1C 24 00 00 00 09 29 2D 22 3B +=01CE
7380- 3F 32 0E 2D 15 36 3B 27 23 00 00 00 09 2D 2D 3F +=021E
7390- 36 36 36 07 00 00 09 36 36 36 29 25 21 24 24 07 +=0212
73A0- 00 00 09 36 36 15 15 0C 0C 24 24 07 00 00 09 36 +=0145
73B0- 36 36 2C 0C 34 15 35 24 24 24 07 00 00 09 2E 2A +=01F6
73C0- 3A 3A 32 24 21 21 21 21 36 12 2B 32 06 00 00 09 +=0202
73D0- 36 29 21 21 36 3B 32 36 0E 00 00 00 09 2D 2D 36 +=0221
73E0- 33 33 33 33 2D 2D 04 00 00 09 2D 2D 3F 3F 36 36 +=0277
73F0- 36 25 24 24 36 36 2E 2D 04 00 00 09 2A 2A 2A 2A +=021F
7400- 32 00 00 00 09 2D 2D 36 36 36 3F 3F 0F 2D 25 24 +=023A
7410- 24 05 00 00 09 12 12 0C 0C 15 15 06 00 00 10 12 +=00C0
7420- 12 0A 29 2D 25 00 00 00 01 00 00 00 09 11 2A 2D +=0109
7430- 32 36 3F 27 23 29 25 00 00 00 09 36 36 36 2C 0E +=0224
7440- 2D 23 21 24 3B 17 06 00 00 09 09 11 11 22 3B 3F +=01BD
7450- 32 36 29 25 21 00 00 00 09 09 09 36 36 36 3C 1E +=01EE
7460- 27 23 24 29 15 06 00 00 09 09 09 12 32 3F 3F 24 +=01B3
7470- 29 35 32 3A 27 23 04 00 00 09 09 29 3A 33 3E 2F +=022D
7480- 2D 3D 37 36 07 00 00 09 12 32 0E 25 21 1C 2F 0D +=01D7
7490- 36 36 1E 3F 07 00 00 09 36 36 36 24 2C 0C 2D 32 +=0236
74A0- 36 06 00 00 09 09 16 2F 36 36 2D 3B 3F 00 00 00 +=01A6
74B0- 09 09 31 3A 35 36 36 3B 0F 1C 07 00 00 09 36 36 +=0200
74C0- 36 2C 0C 0C 0C 16 12 1C 04 00 00 09 29 36 36 36 +=01A2
74D0- 2D 00 00 00 09 12 36 36 0D 24 24 2B 29 32 36 07 +=01CC
74E0- 00 00 09 12 36 36 24 24 2E 0C 2D 32 36 07 00 00 +=01A5
74F0- 09 12 32 36 29 25 21 24 3B 3F 00 00 00 09 12 36 +=01E1
7500- 36 26 24 25 0C 2D 32 1E 3F 00 00 00 09 12 32 0E +=01C8
7510- 25 21 1C 2F 0D 36 36 3E 00 00 00 09 12 36 36 24 +=01F3
7520- 2C 0C 2D 32 00 00 00 09 09 09 12 3F 3F 32 29 2D +=01CA
7530- 32 3B 3F 07 00 00 09 09 36 3F 2D 2D 3F 36 36 21 +=0260
7540- 21 00 00 00 09 12 36 36 29 0C 35 24 24 04 00 00 +=015E
7550- 09 12 36 15 15 0C 0C 24 04 00 00 09 12 36 36 21 +=0163
7560- 21 24 36 2E 22 21 24 04 00 00 11 0A 0E 0E 30 33 +=01AE
7570- 2B 09 21 23 23 21 21 00 00 00 09 12 15 15 15 15 +=014C
7580- 1F 1B 0C 0C 0C 0C 04 00 00 09 12 36 0E 2D 1E 13 +=012B
7590- 2D 0C 24 24 04 00 00 09 12 2D 2D 1E 17 17 17 2D +=018A
75A0- 2D 04 00 00 09 09 29 3F 32 17 0E 36 29 04 00 00 +=0165
75B0- 09 09 36 36 36 06 00 00 09 29 15 2E 32 33 1E 3F +=01F1
75C0- 00 00 FE 09 0C 15 0E 0C 05 00 00 FF 01 00 2D 3A +=02AE
75D0- 3A 32 16 06 00 +=0088
```

zu Abb. 3.2.3.1

Die Shape-Tabelle ist hier als Hex-Dump mit 16 Bytes pro Zeile abgedruckt. Links steht die vierstellige Anfangsadresse jeder Zeile, dann folgen 16 Bytes und schließlich deren 16-Bit-Prüfsumme.

Bei der Besprechung des Apple-Disketten-Betriebssystems DOS 3.3 wird noch näher auf solche Befehle wie PRINT CHR$(4)"BLOAD SHAPETABLE" einge-

```
10REM SHAPETEST
20PRINT:PRINTCHR$(4)"BLOAD SHAPETABLE"
30POKE232,0:POKE233,112:HGR:HCOLOR=3
40SCALE=1:FORI=0TO2:FORJ=0TO31
50DRAWI*32+J+1ATJ*8,I*8:NEXT:NEXT
```

Abb. 3.2.3.2 Testprogramm zum Ausgeben von ASCII-Shapes

gangen werden, ebenso auf den Befehl BSAVE. Vorab sei allerdings hier schon gesagt, wie man die eingetippte Shape-Tabelle mit den ASCII-Zeichen auf Diskette speichern kann:

BSAVE SHAPETABLE,A$7000,L$5D5

Zur Kontrolle der korrekten Eingabe kann man das in *Abb. 3.2.3.3* ebenfalls als Hex-Dump abgedruckte Programm einsetzen. Es liefert genau das hier abgedruckte Hex-Dump-Format mit 16 Bytes pro Zeile und einer Prüfsumme am Zeilenende, wenn man vorher den auszudruckenden Speicherbereich mit BLOAD von Diskette lädt (dann werden nämlich Anfangs- und Endadresse automatisch korrekt gesetzt). Allerdings muß man unbedingt vorher festlegen, wohin die Ausgabe geleitet werden soll:

0812=	Ausgabe auf:
B0	Bildschirm
B1	Slot 1 (meist Drucker)
B2	Slot 2 (im Listing)

Man geht also nach dem Abspeichern des Prüfsummen-Programms mit BSAVE DUMP,A$800,L$BB so vor:

1. Von Basic in den Monitor gehen (CALL-151) und Prüfsummen-Programm von Disk laden: BLOAD DUMP
2. Eventuell Ausgabe-Slot in 0812 anpassen (siehe oben), z.B. 0812:B0 für Bildschirm-Ausgabe
3. Auszudruckendes bzw. zu prüfendes Programm von Disk laden: BLOAD SHAPETABLE
4. Prüfsummen-Programm starten: 800G

```
0800- 20 9E 08 8D 8D C8 C5 D8 C4 D5 CD D0 8D 8D 84 D0 +=09E9
0810- D2 A3 B2 8D 8D 98 AD 72 AA 85 3C AD 73 AA 85 3D +=08EF
0820- A5 3D A6 3C 20 41 F9 A9 00 85 3E 85 3F A9 AD 20 +=06C4
0830- ED FD A2 10 A9 A0 20 ED FD A0 00 B1 3C 20 DA FD +=0973
0840- B1 3C 18 65 3E 85 3E 90 02 E6 3F E6 3C D0 02 E6 +=06FC
0850- 3D AD 60 AA 18 6D 72 AA 85 40 AD 73 AA 6D 61 AA +=079C
0860- C5 3D D0 06 A5 40 C5 3C F0 09 CA D0 C7 20 8B 08 +=07CB
0870- 4C 20 08 20 8B 08 20 9E 08 84 D0 D2 A3 B0 8D 8D +=0680
0880- C6 C5 D2 D4 C9 C7 A1 8D 8D 98 60 20 9E 08 A0 AB +=0985
0890- BD 98 A5 3F A6 3E 20 41 F9 A9 8D 4C ED FD 68 85 +=08D0
08A0- 40 68 85 41 E6 40 D0 02 E6 41 A0 00 B1 40 C9 98 +=077F
08B0- F0 06 20 ED FD 18 90 EC 6C 40 00 +=0540
```

Abb. 3.2.3.3 Programm zur Ausgabe von Speicherbereichen als Hex-Dump mit zeilenweisen Prüfsummen

Wenn nun in einer Zeile eine vom Original-Listing abweichende Prüfsumme ausgegeben wird, so ist ein Byte dieser Zeile falsch und muß korrigiert werden. (Einen kleinen Haken hat die Prüfsummen-Methode allerdings: Vertauschte Bytes ergeben die gleiche Prüfsumme und werden daher nicht als Fehler erkannt.)

Variablentypen

Im Gegensatz zu MBasic gibt es beim Applesoft-Interpreter keine Unterscheidung zwischen Variablen einfacher und doppelter Genauigkeit und daher auch keine Befehle zur Definition und Umwandlung dieser Typen wie DEFDBL, CDBL usw. Statt dessen gibt es nur drei Typen, wobei der Variablenname (hier als Beispiel A) aus beliebig vielen Buchstaben und Ziffern bestehen darf, wovon allerdings nur die ersten beiden Zeichen signifikant sind (ABC ist also gleichwertig mit AB). Ferner muß wie bei MBasic das erste Zeichen ein Buchstabe sein, und Variablen dürfen keine Basic-Befehle (reservierte Worte) enthalten:

A	Fließkomma-Variable mit neunstelliger Mantisse; Wertebereich +/− 9.99999999 E +/− 37
A%	Integer-Variable (Ganzzahl-Variable), −32768...+32767
A$	Stringvariable, Länge bis zu 255 Zeichen

Arrays sind mit allen drei Variablentypen möglich, ein Array darf bis zu 255 Dimensionen umfassen. Eine Unterscheidung von FRE(0) und FRE(X$) gibt es hier nicht; FRE führt grundsätzlich immer eine String-Garbage-Collection durch (vgl. MBasic). In FOR-NEXT-Schleifen dürfen nur Fließkomma-Variablen als „Laufvariable" verwendet werden. Als Zahlenwerte sind ausschließlich dezimale Zahlen zulässig.

Operatoren

Unter Applesoft stehen nicht alle unter MBasic vorhandenen Operatoren zur Verfügung, sondern nur die häufiger gebrauchten:

=	Gleich	NOT	Logische Negation
<>	Ungleich	OR	Logische Oder-Verknüpfung
<	Kleiner	AND	Logische Und-Verknüpfung
>	Größer		
<=	Kleiner oder gleich		
>=	Größer oder gleich		

Ein wesentlicher Unterschied zu MBasic und den meisten anderen Basic-Dialekten ist, daß NOT, AND und OR nicht bitweise durchgeführt werden, sondern allein die logischen Resultate 0 oder 1 liefern. Auch die Verwendung von 1 für „wahr" ist Applesoft-spezifisch; bedingt durch die bitweise Verknüpfung verwenden andere Interpreter meist −1 für „wahr". Eine bitweise Operation wie A = B AND 127 ist unter Applesoft also nicht sinnvoll.

Editieren

Unter Applesoft gibt es keinen EDIT-Befehl, sondern eine Art einfachen bildschirmorientierten Editor für nachträgliche Korrekturen. Hierbei gelten folgende Regeln: Die Taste „Pfeil links" (Hex-Code 08) löscht das zuletzt eingegebene Zeichen im Zeilenpuffer und setzt den Cursor um eine Stelle nach links. „Pfeil rechts" bringt das unter dem Cursor auf dem Bildschirm stehende Zeichen in den Zeilenpuffer und bewegt dann den Cursor eine Stelle nach rechts. Nach einem Druck auf die Escape-Taste kann man den Cursor mit den (im Viereck angeordneten) Tasten J, K, M und I auf dem Bildschirm nach links, rechts, oben und unten bewegen, ohne die so übergangenen Zeichen in den Puffer zu bringen, während jede andere Taste diese Escape-Funktion beendet. Die Return-Taste schließlich überträgt die im Puffer stehende Zeile zum Basic-Interpreter, dabei wird der rechts vom Cursor stehende Rest der Zeile gelöscht.

GW-Basic

Auch GW-Basic wurde von Microsoft entwickelt. Dabei handelt es sich um einen Interpreter für IBM-PC-kompatible Computer auf der Basis der CPU-Typen 8088, 8086, 80186 oder 80286, die mit dem Betriebssystem MS-DOS arbeiten (dazu später mehr). GW-Basic beinhaltet einen bildschirmorientierten Editor, wobei es im Gegensatz zum Apple-II nicht erforderlich ist, mit dem Cursor die gesamte Zeile, die vom Bildschirm in den Interpreter übernommen werden soll, mit dem Cursor abzufahren: Es genügt, den Cursor irgendwo in die entsprechende Zeile zu bewegen und dann Return zu drücken (vgl. Commodore-Basic).

Die folgenden Befehle sind auch in MBasic zu finden und dort bereits erläutert worden:

ABS	AND	ASC	ATN
AUTO	CALL	CHAIN	CHR$
CINT	CLEAR	CLOSE	COMMON
CONT	COS	CSNG	CVD
CVI	CVS	DATA	DEF FN
DEFDBL	DEFINT	DEFSNG	DEFSTR
DELETE	DIM	EDIT	END
EOF	EQV	ERASE	ERL
ERR	ERROR	EXP	FIELD
FILES	FIX	FOR/TO/STEP/NEXT	
FRE	GOSUB	GOTO	HEX$
IF	IMP	INKEY$	INPUT
INPUT#	INPUT$	INSTR	INT
KILL	LEFT$	LEN	LET
LIST	LLIST	LOAD	LOF
LOG	LPOS	LPRINT	LSET
MERGE	MID$	MKD$	MKI$

MKS$	MOD	NAME AS	NEW
NOT	OCT$	OPEN	OPTION BASE
OR	OUT	PEEK	POKE
POS	PRESET	PRINT	PRINT #
PSET	PUT	RANDOMIZE	READ
REM	RENUM	RESET	RESTORE
RESUME	RETURN	RIGHT$	RND
RSET	RUN	SAVE	SGN
SIN	SPACE$	SPC	SQR
STOP	STR$	STRING$	SWAP
SYSTEM	TAB	TAN	TROFF
TRON	USING	USR	VAL
VARPTR	WAIT	WEND	WHILE
WIDTH	WRITE	WRITE #	XOR

Für einige der GW-Basic-Befehle gibt es die Möglichkeit, sie einzugeben, indem man die ALT-Taste zusammen mit einer Buchstabentaste drückt, die gleich dem jeweiligen Anfangsbuchstaben ist: AUTO, BSAVE, COLOR, DELETE, ELSE, FOR, GOTO, HEX$, INPUT, KEY, LOCATE, MERGE, NEXT, OPEN, PRINT, RUN, SCREEN, THEN, USING, VAL, WIDTH, XOR.

Nun aber zu den noch nicht erläuterten GW-Basic-Befehlen, die es in MBasic nicht gibt. Optionale Angaben stehen wieder in spitzen Klammern.

BEEP
Gleichwertig mit PRINT CHR$(7) = Erzeugen eines kurzen Pieptons.

BLOAD N$ <,Offset>
Laden eines Speicherbereichs mit den Namen N$. Ein Offset kann angegeben werden, wenn dies an einer anderen Speicheradresse erfolgen soll, als sie bei BSAVE angegeben wurde.

BSAVE N$,Offset,Länge
Speichern eines Adressenbereichs auf Diskette. Der Offset bezieht sich auf den Beginn des mit DEF SEG gerade aktivierten 64-KByte-Segments.

CHDIR N$
Wechsel des übergeordneten Disk-Directory-Namens, z.B.:
CHDIR "B:RECHENPROGRAMME"

CIRCLE <STEP> (x,y),r<,F<,S,E<,A>>>
Zeichnet einen Kreis mit dem Mittelpunkt x,y, dem Radius r und der Farbe F (0...3). Bei Verwendung von STEP werden nicht absolute Koordinaten, sondern zum zuletzt gezeichneten Punkt relative verwendet. Optional kann man mit S und E Start- und Endwinkel im Bogenmaß angeben, um nur einen Kreisausschnitt zu zeichnen. Mit A (Aspekt, Normalwert 1) kann man je nach Monitor ellipsenförmige Verzerrungen korrigieren. Wird der Endwinkel negativ angegeben, so zeichnet der Rechner eine Linie vom Kreismittelpunkt zum Ende des Kreisbogens.

CLS <n>
Löschen des gesamten Bildschirms (CLS oder CLS 0), des Grafik-Bildteils (CLS 1) oder des Text-Bildteils (CLS 2) und Setzen des Cursors in die linke obere Ecke.

COLOR <TH> <,P> <,GV> <,GH> <,TV>
Setzen der Farben: TH = Text-Hintergrund (0...15), GV = Grafik-Vordergrund, GH = Grafik-Hintergrund, TV = Text-Vordergrund. Mit P = 0 oder 1 kann man zwei Farbpaletten wählen:

Palette	Farbe 1	Farbe 2	Farbe 3
0	Grün	Rot	Gelb
1	Cyan	Magenta	Weiß

Verwendet man COLOR ohne Argumente, dann werden folgende Werte angenommen: Palette = 1, Vordergrund = 3, Hintergrund = 0. Bei hochauflösender Grafik sind nur GV, GH und TV als Argumente zulässig, bei reiner Textdarstellung nur TV und TH, bei mittlerer Auflösung dagegen alle. Im Schwarzweiß-Textmodus sind folgende Werte für den Vordergrund möglich:

0 schwarz
1 unterstrichene weiße Zeichen
2...7 weiß

Der Hintergrund ist hier bei den Werten 0...6 weiß und bei 7 schwarz. Ohne COLOR-Befehl nimmt der Computer zunächst COLOR 7,0 an (also weiße Zeichen auf schwarzem Grund). Manche GW-Basic-Versionen lassen den Befehl bei hochauflösender Grafik nicht zu.

COM(n) ON/OFF/STOP, ON COM(n) GOSUB m
Der Kanal n einer Kommunikations-Schnittstelle (z.B. V.24 für Modem) löst nach COM(n) ON einen Interrupt aus, sobald ein Zeichen empfangen wurde. Das Programm springt dann mit ON COM(n) GOSUB m zur Zeilennummer m, um das Zeichen zu verarbeiten. COM(n) OFF sperrt diesen Interrupt wieder.

CSRLIN
Liefert die Nummer der Bildschirmzeile, in der sich der Cursor gerade befindet. Beispiel: C=CSRLIN.

DATE$
Datum der Echtzeituhr. Zwei Formen sind möglich: A$=DATE$ liefert das Datum im Format „MM-DD-JJ" (Monat, Tag, Jahr, d.h. amerikanische Reihenfolge). DATE$=A$ setzt dagegen die Echtzeituhr auf das in A$ stehende Datum. JJ darf auch vierstellig eingegeben werden.

DEF SEG <= A>
Setzen des CPU-Segmentregisters auf ein bestimmtes 64-KByte-Segment. Wird die Adresse A weggelassen, so wird jene des Basic-Interpreters selbst angenommen.

DRAW G$
Zeichnen des Grafik-Strings G$. Dieser kann folgende Kommando-Buchstaben enthalten:

U n	nach oben	D n	nach unten
L n	nach links	R n	nach rechts
E n	rechts oben	F n	rechts unten
G n	links unten	H n	oben links

Mx,y relative Bewegung ab der letzten Position oder, falls vor x kein Vorzeichen steht, absolute Bewegung

B Ein B vor einem der obigen Kommandos verhindert das Zeichnen, sorgt also für eine „unsichtbare" Bewegung

N Ein N vor einem der obigen Kommandos sorgt dafür, daß die ursprünglichen Koordinaten nach Ausführung wieder hergestellt werden

A n Winkel n setzen (0...3 entsprechend 0, 90, 180, 270 Grad)

TA n Rotation um n Grad (negativ = im Uhrzeigersinn)

C n Farbe n setzen (0...3 bei mittl. Aufl., 0/1 bei hoher Aufl.)

S n Maßstab setzen

X A$ Den String A$ ausführen (ähnlich GOSUB!)

Pn,m Geschlossene Figur mit der Farbe n ausfüllen, m=Grenzfarbe

Bei den Bewegungs-Kommandos kann n auch entfallen; dann wird 1 als Schrittweite angenommen. Beispiel zum Zeichnen eines Rechtecks:

```
SCREEN 1:A=40:DRAW"U=A;R=A;D=A;L=A;"
```

ENVIRON N$
Definieren von N$ als neuen Programm-Umgebungs-String, z.B.
ENVIRON "PATH=A:RECHNUNGEN;A:LAGER"

ENVIRON$ N$
Liefert den Programm-Umgebungs-String des definierten Datenpfades.

ERDEV
Liefert den Fehlercode des zuletzt benutzten Peripheriegeräts als Zahl.

ERDEV$
Liefert nach einem Fehler den Namen des zuletzt benutzten Peripheriegeräts.

ERR, ERL
Nach einem Fehler im Basic-Programm liefert ERR den Fehlercode als Zahl und ERL die Nummer der Zeile, in der der Fehler auftrat. Wenn drr Fehler bei einem Direkt-Kommando auftrat (z.B. LIST), wird ERL = 65535.

GET <#> n,m
Liest m Bytes von der Schnittstellen-Datei n in den Puffer (üblicherweise beim Lesen von Zeichen von der Kommunikations-Schnittstelle verwendet). Beispiel: GET# 1,80. Oder: Lesen eines Records mit der Nummer m von der Disketten-Datei n.

GET <STEP> (x1,y1)– <STEP> (x2,y2),A%
Einlesen des Grafik-Bitmusters auf dem Bildschirm, das rechteckförmig von x1,y1 und x2,y2 begrenzt wird, in eine numerische Array-Variable (hier A%).

IOCTL <#> n,A$
Definition eines Strings A$, der immer dann dem Treiberprogramm übergeben wird, wenn die Datei n geöffnet wird.

IOCTL$ (<#> n>
Liefert den Status des Treiberprogramms von Datei n als String.

KEY OFF/ON/LIST/n A$

Funktionstasten-Befehle; folgende Formate sind möglich:

KEY OFF	löscht die Funktionstasten-Anzeige in der letzten Bildschirm-Zeile.
KEY ON	stellt die Funktionstasten-Belegung in der letzten Bildschirmzeile dar (Normalzustand beim Einschalten). Die Strings werden dabei auf sechs Zeichen begrenzt.
KEY LIST	Listet die Belegung aller Funktionstasten in voller Länge auf dem Bildschirm.
KEY n,A$	Die Funktionstaste n wird mit dem String A$ belegt, der bis zu 15 Zeichen lang sein darf. Soll der String mit einem Return-Zeichen abgeschlossen werden, so geht das mit KEY n,A$+CHR$(13).
KEY(n) ON	Erlaubt den Befehl ON KEY(n) GOSUB
KEY(n) OFF	Sperrt den Befehl ON KEY(n) GOSUB
KEY(n) STOP	Der Interpreter merkt sich den Druck auf Funktionstaste n bis zum nächsten KEY(n) ON
ON KEY(n) GOSUB m	Sprung zur Zeile m, wenn Funktionstaste n gedrückt wird

LCOPY <n>
Bildschirm auf Drucker kopieren, wenn n fehlt oder Null ist. Andernfalls hängt die Wirkung vom installierten Interrupt-5-Treiber ab.

LINE <<STEP> (x1,y1)>-<STEP>(x2,y2) <,<f><,B <F>>><,a>
Zeichnen einer Linie mit absoluten oder, bei Verwendung von STEP, mit relativen Koordinaten. Die B-Option zeichnet ein Rechteck, die F-Option füllt es mit der Farbe f. Die 16-Bit-Integer-Zahl a erlaubt es, ein bestimmtes Bitmuster z.B. für punktierte Linien zu definieren.

LIST <<a>-<b>> <,c>
Die Syntax des LIST-Befehls in GW-Basic ist etwas komfortabler als in MBasic. Nach optionaler Angabe des Zeilennummer-Bereichs kann, durch ein Komma getrennt, auch ein Peripheriegerät oder ein Dateiname angegeben werden. Beispiel zum Listen des Programms auf den Drucker (alternativ zu LLIST):

LIST,"LPT1:"

LOC(n)
Liefert die momentane Position innerhalb der Datei n.

LOCATE <z><,<s><,<c><,<v><,n>>>>
Der Cursor wird an eine bestimmte Stelle auf dem Bildschirm gesetzt. Die Parameter sind: z = Zeile, s = Spalte, c = Cursor-Art (0...10), v/n = Cursor-Zeile (Grafikraster!) von/nach.

MKDIR A$
Gestattet das Eröffnen eines neuen Disk-Inhaltsverzeichnisses mit A$ = Datenpfad, z.B. MKDIR "FINANZEN".

OPEN A$ FOR OUTPUT/INPUT AS <#> n
Für OPEN ist sowohl die MBasic-Syntax (z.B. OPEN"I",1,"NAME") als auch die hier angeführte möglich. Als A$ sind die Namen einer Disketten-Datei oder eines Peripheriegeräts bzw. von Tastatur oder Bildschirm zulässig (inklusive Doppelpunkt!):

KYBD:	Tastatur	SCRN:	Bildschirm
LPTn:	Drucker n (1...3)	COMn:	Schnittstelle n (1...4)

Die LPTn-Geräte (Line Printer) und der Bildschirm dürfen nur mit OUTPUT (Ausgabe) eröffnet werden, die Tastatur nur als INPUT (Eingabe).

OPEN "COM n:g,p,d,s"
Öffnen der Kommunikations-Schnittstelle n für Ein- und Ausgabe mit folgenden Parametern:

g Geschwindigkeit in Baud (75...9600)
p Parity; E=even (gerade), O=odd (ungerade), M=Mark (1), S=Space (0), N=No (keine)
d Anzahl der Datenbits (5...8)
s Anzahl der Stopbits (1; 1.5; 2)

OPEN COM läßt je nach Ausführung der Schnittstelle noch eine Anzahl weiterer Optionen zu.

PAINT <STEP>(x,y)<,ff><,<gf><,hf>>
Ausfüllen einer begrenzten Fläche, Start an den Koordinaten x,y, ff=Füllfarbe, gf=Grenzlinien-Farbe, hf=Hintergrundfarbe. Die STEP-Option erlaubt die Verwendung relativer Koordinaten ab dem zuletzt gezeichneten Punkt.

PAINT (x,y),A$
Füllen einer Fläche um x,y mit dem Bitmuster im String A$. Beispiel für eine gepunktete, grau wirkende Fläche: A$=CHR$(85), wobei dezimal 85 dem Bitmuster der Dualzahl 01010101 entspricht, bzw. A$="U".

PLAY A$
Der String A$ wird als Melodie interpretiert und gespielt. Er kann aus folgenden Zeichen bestehen:

A...G	Noten A...G; für Tonerhöhung ist # oder + voranzustellen, ein Minuszeichen für einen halben Ton tiefer
On	Oktave n wählen (n = 0...6)
>n	Ton n eine Oktave höher spielen
<n	Ton n eine Oktave tiefer spielen
Nn	Ton n = 0...84 spielen (sieben Oktaven möglich)
Pn	Pause der Dauer 1/n mit n = 1...64
Ln	Notenlänge 1/n mit n = 1...64
.	Note verlängern
Tn	Tempo = Zahl der Viertelnoten pro Minute; n = 32...255
MF	Puffer ausschalten (Tonerzeugung im Vordergrund; beim Einschalten voreingestellt)
MB	Puffer einschalten (Tonerzeugung im Hintergrund)
MN	Musik normal: Notenlänge = 7/8 L
ML	Musik legato: Notenlänge = L
MS	Musik staccato: Notenlänge = 3/4 L
X A$	Unter-String A$ ausführen (wie GOSUB!)

Beispiel für eine kurze Melodie:

```
PLAY "T180 O2 P2 P8 L8 GGG L2 E- P24 P8 L8 FFF L2 D"
```

PLAY(0)
Liefert die Zahl der im Hintergrund-Puffer noch verbliebenen Noten.

PLAY ON/OFF/STOP; ON PLAY(n) GOSUB
Nach dem Befehl PLAY ON wird der Unterprogramm-Sprung ON PLAY(n) GOSUB ausgeführt, sobald weniger als n Noten im Hintergrund-Puffer übrig sind. ON PLAY(n) GOSUB wird dann automatisch gesperrt. Das Sperren ist jedoch auch vorher schon mit PLAY OFF möglich.

PMAP (a,b)
Liefert die Fenster-Koordinate f der physikalischen Koordinate p und umgekehrt (kann x- oder y-Koordinate sein). Für b sind folgende Werte möglich:

0	fx zu px	1	fy zu py
2	px zu fx	3	py zu fy

POINT(x,y)
Liefert die Farbe an den Bildschirm-Koordinaten x,y.

POINT(n)
Liefert eine der beiden momentanen Grafik-Koordinaten x und y entweder als physikalische oder als relative Fenster-Werte. Für n sind folgende Zahlen möglich (vgl. PMAP):

0	px	1	py
2	fx	3	fy

PUT(x,y),A% <,v>
Neben der von MBasic her bekannten besitzt PUT wie GET in GW-Basic eine zusätzliche Funktion für Grafiken. Das mit GET in eine numerische Array-Variable gelesene Grafik-Bitmuster wird mit PUT aus A% wieder auf den Bildschirm zurückgeschrieben. Mit der Option V kann gewählt werden, welche Verknüpfung mit dem schon auf dem Schirm vorhandenen Bild erfolgen soll; für V kann man PSET, PRESET, AND, OR oder XOR schreiben. Bei fehlendem v gilt XOR.

RMDIR A$
Löscht das Disk-Directory mit dem Namen A$.

SCREEN(z,s <,v>)
Mit v=0 liefert SCREEN den ASCII-Code an der Bildschirmposition z,s (Zeile, Spalte), mit v=1 den Farbcode des Zeichens.

SCREEN <v> <,<f><,<b><,s>>>
Setzt die Parameter für den Bildschirm:

v=0 Textmodus
v=1 Grafik mit mittl. Auflösung
v=2 Grafik mit hoher Auflösung (nur schwarzweiß)
v=3 Grafik mit superhoher Auflösung (nur schwarzweiß, nicht immer möglich)
f=0 im Textmodus: Farbe aus, bei mittl. Auflösung: Farbe ein
f=1 im Textmodus: Farbe ein, bei mittl. Auflösung: Farbe aus
b=0...3: Wahl der Ausgabe-Seite, 80 Zeichen pro Zeile
b=0...7: Wahl der Ausgabe-Seite, 40 Zeichen pro Zeile
s=0...3: Wahl der sichtbaren Seite, 80 Zeichen pro Zeile
s=0...7: Wahl der sichtbaren Seite, 40 Zeichen pro Zeile

Mit den Parametern s und b ist es also beispielsweise möglich, während der Anzeige der Video-RAM-Seite 0 in die Seite 1 „blind" zu schreiben und letztere erst dann anzuzeigen, wenn die Grafik auf ihr fertig gezeichnet ist.

SHELL <A$>
Maschinenprogramm namens A$ von Disk laden und ausführen, dann zu GW-Basic zurückkehren. Wenn der Name A$ fehlt, gelangt man zum Betriebssystem MS-DOS und darf alle DOS-Befehle verwenden, die in der Datei COMMAND stehen. Mit EXIT gelangt man dann zu GW-Basic zurück.

SOUND f,d
Ein Ton der in f angegebenen Frequenz (Hertz) und der Dauer d (eine Einheit = 0,055 s) wird erzeugt.

TIME$
Zwei Schreibweisen sind möglich: A$=TIME$ liest die Uhrzeit von der 24-Stunden-Echtzeituhr in den String A$, das Format ist „hh:mm:ss" (Stunden, Minuten, Sekunden). TIME$=A$ setzt die Echtzeituhr auf den in A$ angegebenen Wert.

TIMER
Liefert die Zeit seit Mitternacht in hundertstel Sekunden.

TIMER ON/OFF/STOP; ON TIMER(n) GOSUB
Mit TIMER ON wird der Unterprogrammsprung ON TIMER GOSUB freigegeben, soband n Sekunden abgelaufen sind. TIMER STOP verhindert diesen Sprung vorübergehend bis zum nächsten Mal. TIMER OFF schaltet diesen Mechanismus ganz ab.

VARPTR$(v)
Neben den von MBasic bekannten Befehlen VARPTR und VARPTR# gibt es in GW-Basic VARPTR$, um einen 3-Byte-String als Information über die Variable v zu liefern:

Byte 1: 2 = Integer, 3 = String, 4 = einf. genau, 8 = doppelt genau
Byte 2: niederwertiges Byte der Adresse
Byte 3: höherwertiges Byte der Adresse

VIEW <b>(px1,py1)−(px2,py2) <,f> <,g>
Definiert die physikalischen Bildschirm-Grenzen für grafische Befehle mit px, py als Koordinaten eines Rechtecks, f als Hintergrundfarbe und g als Farbe der Grenzlinien.

VIEW PRINT <a TO b>
Definition des Textfensters im Bildschirm mit a,b als oberste und unterste Zeile. Ohne a und b wird der gesamte Schirm als Textfenster betrachtet.

WIDTH <LPRINT> n oder WIDTH m,n oder WIDTH pg,n
Definieren der Zeilenbreite = n für Ausgaben auf den Drucker (LPRINT-Option), auf die Datei m oder auf das Peripheriegerät pg. Im Gegensatz zu MBasic kann mit WIDTH SCRN:,40 oder WIDTH 40 der Bildschirm von 80 auf 40 Zeichen pro Zeile umgeschaltet werden.

WINDOW <<SCREEN> (fx1,fy1)−(fx2,fy2)>
Setzt die logische Größe eines Grafik-Fensters mit fx,fy als Fenster-Koordinaten. Die SCREEN-Option dreht die y-Koordinate um, so daß fx1,fy1 oben links und fx2,fy2 unten rechts sind.

Variablen

Die Schreibweise von Variablennamen und den einzelnen Variablentypen ist unter GW-Basic identisch mit MBasic. Bis zu 40 Zeichen des Namens sind signifikant. Ein nicht besonders gekennzeichneter Typ wird als Fließkomma-Variable mit einfacher Genauigkeit angenommen. Wie in MBasic arbeiten logische Operatoren bitweise. Als Wert für „wahr" wird −1 verwendet, für „falsch" Null.

Tastenfunktionen

GW-Basic setzt eine zum IBM-PC kompatible Tastatur voraus. Steuerfunktionen werden meist durch gleichzeitiges Drücken der CTRL-Taste und einer weiteren Taste erreicht:

CTRL-Break	a) Basic-Programm unterbrechen b) Editieren eine Zeile abbrechen c) AUTO-Funktion beenden
CTRL-Numlock	Ausgabe bis zum nächsten Tastendruck anhalten
CTRL-T	Wechsel der Funktionstasten-Anzeige
CTRL-ALT-DEL	System-Reset (Betriebssystem neu laden)
CTRL-PRTSC	Drucker-Protokollierung von Bildschirm-Ausgaben ein/aus
Shift-PRTSCR	Bildschirminhalt auf den Drucker kopieren
CTRL-L	Bildschirm löschen
HOME	Cursor in die linke obere Ecke setzen
CTRL-HOME	Bildschirm löschen
END	Cursor ans Ende der Zeile (Taste, nicht Basic-Befehl!)
CTRL-END	Rest der Zeile löschen
INS	Insert-Modus ein/aus (Zeichen an Cursorpos. einfügen)
DEL	Zeichen unter dem Cursor löschen
CTRL-CR	Line-Feed-Zeichen erzeugen
ESC	Zeile auf dem Bildschirm löschen (nicht im Speicher!)

Wie schon eingangs erwähnt, besitzt GW-Basic einen bildschirmorientierten Editor. Der Cursor läßt sich mit den Pfeiltasten jeweils um ein Zeichen nach links, rechts, oben oder unten frei auf dem Bildschirm bewegen. Hält man zusätzlich zu den Pfeiltasten „Links" oder „Rechts" die CTRL-Taste gedrückt, so wird genau ein Wort übersprungen (d.h. der Cursor wird bis zum nächsten Leerraum bewegt).

Commodore-Basic

Im Vergleich zu MBasic ist das von Commodore bei den Computern der CBM-Serie sowie beim C-64 eingesetzte Basic eher spartanisch. (Die neueren Geräte C-16 und C-116 besitzen jedoch einen demgegenüber erweiterten Befehlssatz.)

Die von Commodore auf den Markt gebrachten Geräte besitzen durchwegs einen Bildschirm-Editor, d.h. man kann den Cursor frei auf dem Schirm bewegen und Korrekturen durchführen. Wie bei GW-Basic ist es (im Gegensatz zum Apple-II) nicht nötig, mit dem Cursor bis ans Ende einer Bildschirmzeile zu fahren, wenn sie mit Return in den Speicher übernommen werden soll. Im Gegensatz zu MBasic und GW-Basic müssen wie beim Apple-II die einzelnen Befehle bei der Programmeingabe voneinander nicht durch Leerräume getrennt werden.

Wie bei MBasic und GW-Basic sind auch bei den Commodore-Rechnern einige Disketten-Befehle vorhanden. Allerdings wird ein Floppy-Laufwerk weitgehend als normales Peripheriegerät betrachtet, das über den IEC-Bus-Anschluß mit den hierfür vorgesehenen Befehlen angesprochen wird. Kommen wir also zum Befehlssatz des Commodore-Basic. Auch hier sind optionale Angaben durch spitze Klammern gekennzeichnet. Befehle, die nicht auf C-64 und CBM-3032 (Basic 3), sondern nur auf den Geräten CBM-40XX und 80XX zur Verfügung stehen, sind mit

„Basic 4" gekennzeichnet. Zunächst aber zu den Befehlen, die bereits von MBasic her bekannt sind.

MBasic-verwandte Kommandos und Befehle

CONT	DATA	DEF FN	DIM
END	FOR/TO/NEXT/STEP		GOSUB
GOTO	IF/THEN	INPUT	INPUT #
LET	LIST	NEW	ON
POKE	PRINT	PRINT #	READ
REM	RESTORE	RETURN	RUN
STOP	WAIT		

MBasic-verwandte Funktionen

ABS	ASC	ATN	CHR$
COS	EXP	FRE(0)	INT
LEFT$	LEN	LOG	MID$
PEEK	POS	RIGHT$	RND
SGN	SIN	SPC	SQR
STR$	TAB	TAN	USR
VAL			

Hier gibt es trotz gleicher Schreibweise allerdings gewisse Unterschiede zu anderen Interpretern: Ein INPUT-Befehl gibt grundsätzlich ein Fragezeichen aus, auch dann, wenn ein Prompting-Test nach INPUT steht (z.B. INPUT"ANZAHL";A). Mit USR kann man nur eine einzige Maschinenroutine aufrufen, weil nach USR keine Nummer für die Routine stehen darf, sondern lediglich Parameter, die übergeben werden sollen. Nach RESTORE darf wie beim Apple-II keine Zeilennummer stehen. Nach RUN kann kein Programmname angegeben werden, es wird also stets nur das gerade im Speicher stehende Programm gestartet. Der Befehl FRE führt auch bei numerischem Dummy-Argument wie beim Apple-II grundsätzlich immer eine String-Garbage-Collection durch.

Befehle und Kommandos bei Commodore-Basic

APPEND # m,N$ <,Dn> <ON a> (Basic 4)
Öffnet eine auf der Diskette befindliche sequentielle Datei, um sie zu verlängern: Eine folgende PRINT # m-Anweisung schreibt ab dem bisherigen Dateiende neue Daten auf die Datei. N$ enthält den Dateinamen, optional ist ihm (durch ein Komma getrennt) die Laufwerksnummer nachgestellt. Falls die Floppy nicht die IEC-Bus-Adresse 8 besitzt, kann eine andere angegeben werden (a). Beispiel:

APPEND # 1,"0:ADRESSEN"

BACKUP Dn TO Dm <ON a> (Basic 4)
Duplizieren einer ganzen Diskette von einem Laufwerk (n) auf das andere (m). Falls die Floppy nicht die IEC-Bus-Adresse 8 besitzt, kann eine andere angegeben werden (a). Beispiel:

BACKUP D0 TO D1

CLOSE <n>
Schließen der Datei mit der Nummer n. Ohne Angabe von n werden alle Dateien geschlossen; in Basic 4 erscheint eine Fehlermeldung, wenn hierbei keine Datei geöffnet ist.

CLR
Entspricht dem CLEAR-Befehl in MBasic.

CMD n
Nach CMD n werden alle Ausgaben vom Bildschirm auf die Datei n umgeleitet, also z.B. auf einen Drucker oder auf Diskette. Eine Leeranweisung PRINT # n beendet diesen Modus.

COLLECT <Dn> <ON a> (nur Basic 4)
Löschen aller Blöcke auf der Diskette, die nicht von im Inhaltsverzeichnis genannten Daten belegt sind.

CONCAT <Dn,> N$ TO <Dm,> M$ <ON a> (nur Basic 4)
Die beiden sequentiellen Dateien namens N$ und M$ werden verkettet. Beispiel:
CONCAT "ADR1" TO "ADR2"

COPY <Dn> <N$> TO <Dm> <M$> <ON a> (nur Basic 4)
Werden die Namen N$ und M$ angegeben, so kopiert der Befehl die Datei N$ von Laufwerk m mit dem neuen Namen M$ auf Laufwerk m. Fehlen n und m, wird dafür 0 angenommen. Fehlen die Namen, werden alle Dateien kopiert. Beispiel:
COPY D0 TO D1

DCLOSE n <ON a> (nur Basic 4)
Schließen der Disk-Datei n. Mit a kann eine andere IEC-Bus-Adresse als 8 angegeben werden.

DIRECTORY <Dn> <ON a> (nur Basic 4)
Anzeigen des Disk-Inhaltsverzeichnisses auf dem Bildschirm, ohne ein im Speicher stehendes Programm dabei zu zerstören. Alternativ ist auch CATALOG möglich. In Basic 3 läßt sich das Inhaltsverzeichnis so anzeigen:
LOAD "$n" mit n = Laufwerk (0 oder 1, kann entfallen)
Dabei wird allerdings das Programm im Speicher überschrieben.

DLOAD N$ <,Dn> <ON a> (Basic 4)
Laden eines Basic-Programms von Diskette. Beispiel:
DLOAD "PGM1",D0

DOPEN m,N$ <1> <,Dn> <,R> <,W> <ON a> (Basic 4)
Öffnen einer Disk-Datei für Schreib- oder Leseoperationen. Mit 1 kann man die Aufzeichnungslänge angeben; fehlt 1, so wird eine sequentielle Datei angenommen. Hierbei muß R für Read (Lesen) oder W für Write (Schreiben) angegeben sein.

GET A$
Ähnlich INKEY$ in MBasic; Lesen eines Zeichens von der Tastatur bzw. Abliefern eines leeren Strings in A$, wenn keine Taste gedrückt wird. Im Gegensatz zum Apple-II wartet dieser Befehl also nicht auf das Drücken einer Taste. Beispiel:

```
100 GET A$:IF A$="" THEN 100
```

GET # n, A$
Lesen eines Zeichens von der Datei n. Dieser Befehl wartet, bis tatsächlich ein Zeichen anliegt.

HEADER D$, Dn <ON a> (Basic 4)
Formatieren einer fabrikneuen (oder zu löschenden) Diskette mit dem Namen D$. Beispiel:

```
HEADER "DISK1", D0
```

LOAD <N$> <,a> <,sa>
Laden eines Programms von Kassette oder, falls eine IEC-Bus-Adresse a angegeben ist, von einem anderen Peripheriegerät. Der Name N$ kann beim Kassettenrecorder auch weggelassen werden, es wird dann das nächste Programm der Kassette geladen. Gibt man als Sekundäradresse sa den Wert 1 an, so wird das Programm nicht ab der normalen Basic-RAM-Startadresse geladen, sondern ab dort, von wo es aufgezeichnet wurde (z. B. Maschinenprogramme). Beispiel:

```
LOAD "FAKTURA", 8
```

OPEN n <,a> <,sa> <,N$>
Öffnen einer Datei mit der Nummer n auf der IEC-Bus-Adresse 8 mit der optionalen Sekundäradresse sa. Bei Disketten-Operationen muß der Name N$ zusätzlich, durch Kommata getrennt, den Dateityp (SEQ = sequentiell, U = wahlfrei, P = Programmdatei) und die Zugriffsart (R = Read, W = Write) enthalten. Vor dem Namen kann die Laufwerksnummer stehen. Beispiel:

```
OPEN 1,8,2,"1:DATEN,SEQ,R"
```

RENAME <Dn> N$ TO M$ <ON a> (Basic 4)
Die Disk-Datei namens N$ wird in M$ umbenannt. Beispiel:

```
RENAME "ALT" TO "NEU"
```

SAVE <N$> <,a> <,sa>
Speichern des Basic-Programms im Speicher auf Kassette oder, wenn eine IEC-Bus-Adresse a angegeben ist, auf das entsprechende Peripheriegerät. Fehlt die Sekundäradresse oder ist sie Null, so wird nach dem Programm keine zusätzliche Datei-Ende-Markierung aufgezeichnet, was sich praktisch aber nicht auswirkt. Steht vor dem Programmnamen (durch einen Doppelpunkt getrennt) ein „Klammeraffe", so kann eine Datei gleichen Namens überschrieben werden.

SCRATCH <Dn,> N$ <ON a> (Basic 4)
Löschen der Datei namens N$ auf dem Disketten-Laufwerk n. Falls dessen IEC-Bus-Adresse nicht 8 ist, kann sie angegeben werden (a).

VERIFY <N$> <,a>
Vergleich eines Programms auf Diskette mit dem im Speicher stehenden. Beispiel:
VERIFY "PGM"

Funktionen im Commodore-Basic

DS (Basic 4)
Liefert einen numerischen Fehlercode nach Diskettenoperationen. Werte bis 20 bedeuten, daß kein Fehler aufgetreten ist. Liefert DS den Wert 1, so wurde eine Datei gelöscht. Beispiel:

```
IF DS>20 THEN PRINT"FEHLER:";DS
```

DS$ (Basic 4)
Liefert eine Fehlermeldung nach Disketten-Operationen im Klartext. Beispiel:

```
IF DS>20 THEN PRINT DS$
```

POS(0)
Liefert die augenblickliche horizontale Cursor-Position (Spalte). Das Dummy-Argument in Klammern wird ignoriert. Beispiel:

```
PRINT TAB(23);POS(0)
```

ST
Liefert den Status des letzten Zugriffs auf ein externes Gerät (z.B. Kassettenrecorder, Floppy). Ist ST Null, trat kein Fehler auf. Beispiel:

```
IF ST>0 THEN PRINT"FEHLER!"
```

TI
Die Variable TI zählt einen Zeittakt in Schritten von jeweils einer sechzigstel Sekunde (Interrupt-gesteuert). TI kann nur gelesen werden, das Rücksetzen kann über TI$ erfolgen.

TI$
Die Stringvariable TI$ liefert die momentane Uhrzeit im Format hhmmss. Beim Einschalten des Computers muß sie neu gesetzt werden, da die Zeit nicht von einer batteriegepufferten Echtzeituhr, sondern von einem zyklischen Timer-Interrupt geliefert wird. Beispiel:

```
TI$="105500"
PRINT TI$
```

Variablen

Die Variablendarstellung in den CBM-Computern und im C-64 ist mit der im Apple-II identisch und braucht daher nicht nochmals dargestellt zu werden.

Editierfunktionen

Es wurde bereits erwähnt, daß die Commodore-Computer einen eingebauten bildschirm-orientierten Editor besitzen, der ein sehr komfortables Korrigieren von

Programmzeilen ermöglicht. Die Computer der Serie CBM-80XX mit 80 Zeichen pro Zeile bieten daneben folgende Editier-Sonderfunktionen:

CHR$(7) oder Tasten ESC+RVS+g
Kurzer Piepton; wird beim Überschreiten von 75 Zeichen in einer Zeile automatisch erzeugt.

CHR$(14)
Die Darstellung wird von Grafik-Sonderzeichen auf normalen Text mit Groß- und Kleinschreibung zurückgeschaltet.

CHR$(15)
Linke obere Ecke eines Bildschirm-Fensters markieren (Cursor-Position).

CHR$(21) oder Tasten ESC+RVS+u
Löschen einer Zeile auf dem Schirm; die darunterliegenden rücken um eine Zeile nach oben.

CHR$(22) oder Tasten ESC+RVS+v
Die rechts vom Cursor stehenden Zeichen der Zeile werden gelöscht.

CHR$(25) oder Tasten ESC+RVS+q
Alle Bildschirmzeilen werden um eine nach oben verschoben, unten wird dadurch Platz frei.

CHR$(149) oder Tasten ESC+RVS+m
Einfügen einer Bildschirmzeile. Der darunter liegende Text wird nach unten geschoben und macht dadurch Platz an der Cursor-Position frei.

CHR$(143)
Rechte untere Ecke eines Fensters auf dem Bildschirm markieren (Cursor-Position).

CHR$(150) oder Tasten ESC+RVS+V
Löschen des links vom Cursor stehenden Teils einer Bildschirmzeile.

CHR$(153) oder Tasten ESC+RVS+Q
Alle Bildschirmzeilen werden um eine nach unten geschoben, am oberen Bildschirmrand wird dadurch Platz frei.

CHR$(142) oder Tasten ESC+RVS+N
Umschalten der Darstellung auf grafische Sonderzeichen. Kleinbuchstaben werden dabei durch Großbuchstaben ersetzt, Großbuchstaben durch Grafik-Symbole. Das Rückschalten erfolgt mit PRINT CHR$(14).

Man beachte, daß alle diese Funktionen nur auf dem Bildschirm wirksam sind; das im Speicher stehende Programm bleibt solange unverändert, bis man eine Bildschirmzeile durch Druck auf die Return-Taste in den Speicher übernimmt. Steuerzeichen werden übrigens bei allen CBM-Versionen und beim C-64 als inverse Grafik-Symbole dargestellt, wenn sie zwischen zwei Anführungszeichen einge-

schlossen sind. Es ist so auch möglich, sie in PRINT-Anweisungen aufzunehmen. Der interne Zeichencode entspricht allerdings nicht dem ASCII-Standard; vielmehr sind im normalen Text-Darstellungs-Modus Groß- und Kleinschreibung gegenüber ASCII vertauscht.

Der Compiler TASC

TASC ist ein von Microsoft für den Apple-II entwickelter Basic-Compiler. Er verwendet den Applesoft-Interpreter als Editor zur Eingabe und zur Korrektur von Quellprogrammen; ferner kann man mit dem Interpreter auch erste Tests des Programms durchführen. Nach dem Compilieren wird ein Maschinenprogramm erzeugt, das deutlich schneller abgearbeitet werden kann als ein interpretiertes Programm.

Das mit dem Interpreter eingegebene Quellprogramm muß zunächst auf Diskette gespeichert werden, bevor TASC gestartet wird. Der Compiler holt sich nämlich den Quellcode stets von Diskette und generiert auch den Objektcode wieder auf Diskette. Dadurch ist der Compiler-Vorgang zwar nicht allzu schnell, andererseits kann man aber fast beliebig lange Basic-Programme bearbeiten.

TASC läßt gegenüber dem Applesoft-Interpreter die folgenden neuen, zusätzlichen Befehle zu:

REM!INTEGER
Damit kann man eine Variable als Ganzzahl-Variable definieren (wie mit DEFINT in MBasic). Im Gegensatz zum Interpreter sind Integer-Variable auch für FOR-NEXT-Schleifen zulässig. Ferner besitzt TASC eigene Integer-Arithmetik-Routinen, die wesentlich schneller als die Fließkomma-Routinen arbeiten. Die Verwendung von Integer-Variablen ist also immer empfehlenswert, wenn nur mit ganzen Zahlen gerechnet werden soll. Beispiel:

```
REM!INTEGER I,J,K,B(3,7)
```

REM!DEFCOMMON
Definieren von Variablen, die von einem mit BRUN nachzuladenden Programm übernommen werden sollen. Beispiel:

```
REM!DEFCOMMON A,B,C
```

REM!CLEARCHAIN
Freigabe des für DEFCOMMON und USECOMMON benötigten Speicherbereichs; muß vor BRUN am Ende des Programms stehen, an dessen Anfang DEFCOMMON verwendet wurde.

REM!USECOMMON
Übernahme von Variablen von einem vorher ausgeführten TASC-Objektprogramm, das sie mit DEFCOMMON übergeben hat. Beispiel:

```
REM!USECOMMON A,B,C
```

REM!CLEARCOMMON
Der Speicherbereich, der zuvor mit DEFCOMMON zur Variablen-Übergabe vom vorher ausgeführten Programm reserviert wurde, wird wieder als Arbeitsspeicher zugeteilt.

Eine Reihe von Applesoft-Befehlen werden vom TASC-Compiler nicht akzeptiert. Diese sind:

CONT	DEL	LIST	LOAD
LOMEM:	NOTRACE	RECALL	SAVE
SHLOAD	STORE	TRACE	&

Der Befehl „&", der beim Interpreter zum Aufruf eines Maschinensprache-Programms dient, ist bei TASC dafür reserviert, ein einmal gestartetes Objektcode-Programm nach dessen Beendigung erneut laufen zu lassen.

Der Befehl DEFFN kann beim Interpreter innerhalb eines Programms dieselbe Funktion mehrfach umdefinieren, dies muß jedoch stets vor dem Funktionsaufruf mit FN geschehen. Dagegen darf eine Funktion unter TASC nur ein einziges Mal mit DEF FN definiert werden, dies kann aber an beliebiger Stelle des Programms erfolgen.

Die DIM-Anweisung darf bei TASC keine arithmetischen Ausdrücke enthalten; vielmehr muß eine Dimensionierung stets mit ganzen Zahlen erfolgen. Dafür ist es unter TASC möglich, ein Array innerhalb des Programms mehrfach umzudimensionieren, was beim Interpreter zu einer Fehlermeldung führen würde. Fehlt eine DIM-Anweisung, nimmt der Compiler wie der Interpreter 10 als maximalen Index an.

Während schließlich beim Interpreter das Drücken einer nicht-numerischen Taste bei einer numerischen Variablen nach dem GET-Befehl zu einer Fehlermeldung und somit zum Programmabbruch führt, wird dieses Problem vom Compiler abgefangen; er liefert dann den Wert Null.

Die Tastenkombination CTRL-C, mit der man beim Interpreter ein laufendes Programm anhalten kann, funktioniert bei einem compilierten nur während des Abarbeitens eines INPUT-Befehls. Wird ein Objektprogramm so unterbrochen (CTRL-C, Return), so korrigiert das Laufzeitpaket automatisch die Belegung der Zero-Page (Adressenbereich 0000...00FF) für den Interpreter. Das geschieht auch bei den Befehlen END, STOP und NEW.

Ist das Quellprogramm fertig, muß es auf Diskette abgespeichert werden, denn TASC holt den Quellcode von Diskette und schreibt das erzeugte Maschinenprogramm (den Objektcode) ebenfalls auf Diskette. Dadurch dauert der Compilier-Vorgang zwar relativ lang, andererseits ist die Länge der so verarbeitbaren Basic-Programme fast unbegrenzt, weil nur der Compiler selbst im Arbeitsspeicher ste-

hen muß. Übrigens wurde TASC selbst von Microsoft in mehreren Basic-Teilprogrammen geschrieben und compiliert. Der Start des Compilers erfolgt so:

```
RUN TASC
```

Nun wird ein Basic-Programm von der Compiler-Disk gestartet, das seinerseits ein Maschinenprogramm aufruft, das den eigentlichen Compiler enthält. Der folgende Dialog sieht, wenn das zu compilierende Quellprogramm SAMPLE heißt, beispielsweise so aus:

```
SOURCE FILE? SAMPLE                 Name des Quellprogramms
OBJECT CODE FILE:                   Der Compiler schlägt als Name für den
(DEFAULT SAMPLE.OBJ)?               Objektcode SAMPLE.OBJ vor (Return)
MEMORY USAGE:                       Return übernimmt die vom Compiler
DEFAULT CONFIGURATION?              vorgeschlagene Speicherbelegung
OPTIONS:                            Mit Return verzichtet der Benutzer auf
DEFAULT CONFIGURATION?              besondere Optionen
BEGINNING PASS2                     Jetzt wird das Quellprogramm compiliert
CODE GENERATION COMPLETE            Der Objektcode steht nun auf der Disk
COMPILATION INFORMATION             Bei YES statt NO wird eine Tabelle aus-
AND LINE NUMBER REFERENCE           gegeben, welche Zeilennummer welcher
TABLE? NO                           Objektcode-Adresse entspricht
COMPILATION COMPLETE
```

Gibt man bei der Frage OPTIONS als Antwort YES ein, so fragt TASC zusätzlich noch folgendes, wobei die Antwort immer YES oder NO lauten kann:

Compiler fragt:	Normalwert:
COMPILATION LISTING?	YES
PAUSE ON ERRORS?	YES
INTEGER ARITHMETIC?	YES
INTEGER CONSTANTS?	YES
RESUME/DEBUG CODE?	NO

Der erzeugte Objektcode ist allein nicht lauffähig, sondern benötigt im Arbeitsspeicher zusätzlich ein 4 KByte großes Laufzeit-Paket (Runtime Unit), das vorher mit BLOAD RUNTIME von Disk geladen werden muß:

```
BLOAD RUNTIME     Laufzeit-Paket laden
BRUN SAMPLE.OBJ   Compilierten Objektcode laden und starten
```

Mit TASC compilierte Programme laufen im Mittel etwa dreimal schneller als bei Verwendung des Applesoft-Interpreters. Allerdings kann dies nur eine sehr grobe Aussage darstellen. Der Geschwindigkeitsvorteil des Compilers ist besonders eklatant, wenn ausgiebig von der beim Interpreter nur scheinbar vorhandenen Integer-Arithmetik Gebrauch gemacht wird. Bei rechenintensiver Fließkomma-Arithmetik schrumpft der Vorsprung des Compilers allerdings drastisch.

Neben der höheren Arbeitsgeschwindigkeit ist ein Vorteil compilierter Programme auch, daß sie von Unbefugten nicht so leicht geändert werden können (z.B. Entfernen von Urheberrechts-Vermerken), solange der Software-Käufer nur den Objektcode und nicht den Applesoft-Sourcecode erhält. Um Software-Häusern den Verkauf von compilierten Programmen zu ermöglichen, verzichtete Microsoft auf Lizenzgebühren für die Weitergabe des unabdingbaren TASC-Laufzeitpakets an Dritte.

Der Quickbasic-Compiler

Quickbasic ist ein von Microsoft entwickelter Basic-Compiler, der weitgehend mit dem bereits besprochenen GW-Basic-Interpreter bzw. IBM-BASICA kompatibel ist. Quickbasic wird jedoch im Gegensatz zum Interpreter nicht in einer speziell für einen bestimmten Rechnertyp angepaßten Version, sondern systemunabhängig geliefert. Dies hat zur Folge, daß bestimmte systemspezifische Basic-Befehle (z.B. die Grafik 640 × 400 oder eine XOR-Verknüpfung bei Grafik-Befehlen) nicht anwendbar sind.

Bei Verwendung des Quickbasic-Compilers darf Stringlänge bis zu 32767 Zeichen (beim Interpreter nur 255 Zeichen) betragen. Als Sprungziele sind auch alphanumerische Labels statt Zeilennummern möglich, nach THEN jedoch nur mit THEN GOTO.

Zur Steuerung des Compilier-Vorgangs gibt es folgende Metakommandos, die REM-Anweisungen nachzustellen sind (z.B. REM $DYNAMIC):

$LIST +/−	Source-Listing ein (+) oder aus (−)
$OCODE +/−	Objektcode-Listing ein (+) oder aus (−)
$TITLE:titel	Listing-Titel
$SUBTITLE:titel	Listing-Untertitel
$LINESIZE:n	Listing-Zeilenlänge
$PAGESIZE:n	Listing-Seitenlänge
$PAGE	Neue Listing-Seite
$PAGEIF:n	Neue Seite, wenn max. n Zeilen übrig
$SKIP<:n>	Vorschub bis Seitenende oder n Zeilen
$INCLUDE:'name'	Source-File „name" einbinden
$STATIC	Statische Arrays (kein REDIM möglich)
$DYNAMIC	Dynamische Arrays (ERASE/DIM und REDIM möglich)
$MODULE	Modulname für Linker ändern

Zum Compilieren der Datei BEISPIEL.BAS, die im ASCII-Format vorliegen muß (Abspeichern vom Interpreter aus also mit SAVE "BEISPIEL",A), gibt man ein:

```
BASCOM BEISPIEL ; ;
```

Die beiden Strichpunkte bewirken, daß der Compiler nicht mehr nach den Filenamen für Library und Objektcode fragt. Nach dem Source-Filenamen kann man noch folgende „Switches" eingeben:

/A Disassemblierten Code in Source-Listing einbauen
/D CTRL-Break und Fehlerbedingungen zur Laufzeit testen
/E RESUME Zeilennummer kommt vor
/O BCOM10.LIB statt BRUN10.LIB verwenden
/R Arrays reihen- statt spaltenweise ordnen
/S In "" stehende Strings in eigenes File compilieren
/V Befehlsweiser Test bei ON PEN/STRIG/COM/TIMER
/W Zeilenweiser Test bei ON PEN/STRIG/COM/TIMER
/X RESUME kommt ohne Zeilennummer vor

Normalerweise wird als Laufzeitpaket die (getrennte) Datei BRUN10.EXE verwendet; sie wird vom compilierten Programm automatisch geladen. Der Compiler berücksichtigt dies durch Verwenden der Datei BRUN10.LIB. Alternativ hat man auch die Möglichkeit, das Laufzeitpaket BCOM10.LIB in das compilierte Programm einzubinden, so daß das Endprodukt nur noch eine einzige Datei ist. Wenn sich auf einer Diskette mehrere compilierte Programme befinden, ist BRUN10.LIB jedoch günstiger, weil das Laufzeitpaket dann nur einmal (als eigene Datei) auf der Diskette zu stehen braucht, während es bei BCOM10.LIB versteckt einmal in jedem Programm enthalten ist.

Die vom Compiler gelieferte Datei BEISPIEL.OBJ muß nun mit dem MS-DOS-Dienstprogramm LINK (siehe Kapitel über MS-DOS) in eine ausführbare Datei BEISPIEL.EXE konvertiert werden.

Wie schon erwähnt, ist der Quickbasic-Compiler weitgehend kompatibel mit dem GW-Basic-Interpreter. Einige kleine Unterschiede gibt es aber doch:

CALL name,liste	Aufruf eines SUB-Programms
CALLS name<,liste>	Übergibt segmentierte Adressen in Liste
CHAIN "name"	ALL, MERGE, DELETE nicht zulässig
COMMAND$	Liefert Kommandozeile bei Programmaufruf
COMMON <SHARED> liste	SHARED ist für Übergabe an SUB nötig
DEF FNname<(liste)>=Definition <END DEF>	Mehrzeilig möglich
DRAW "string"	DRAW"XA$" als DRAW"X"+VARPTR$(A$) schreiben
FOR.TO.STEP.NEXT	Auch doppelt genaue Laufvariable möglich
LBOUND (A<,d>)	Liefert Basis des Arrays A für Dimension d
LOCK#n	Sperrt File n für externen Zugriff
UNLOCK#n	Gibt File n wieder frei
PLAY "string"	Siehe Einschränkung bei DRAW
REDIM <SHARED>a(n)	Dynamisches Array a(n) neu dimensionieren
RESUME NEXT	Statt NEXT besser Zeilennummer (schneller)
SHARED list	Gemeinsame Variablen-Liste für Subprogramm
STATIC liste	Einfache Variable oder Arrays als lokal definiert
SUB name,liste STATIC	Subprogramm-Definition; nur STATIC möglich
END SUB	Ende eines Subprogramms

UBOUND(A<,n>)	Oberes Limit des Index für Array A (Dim. n)
WIDTH#n,size	Ausgabe auf File n mit size Zeilenbreite

Mit Quickbasic compilierte Programme arbeiten im Schnitt etwa fünfmal schneller als interpretierte. Vorteilhaft ist es (wie bei allen Compilern), möglichst viele Variablen als Integer-Typen zu deklarieren; dies erhöht die Geschwindigkeit drastisch gegenüber Fließkomma-Variablen.

Beispielprogramme in Basic

Die zwei im folgenden besprochenen Programme wurden beide in Applesoft-Basic geschrieben. Das erste realisiert eine kleine Datenbank, das zweite zeigt die Funktionsweise selbstlernender Programme.

Datenbank

Das Datenbank-Programm in *Abb. 3.2.3.4* erlaubt in den Einträgen ein schreibweisen-tolerantes Suchen. Das bedeutet, daß die Begriffe, mit denen bestimmte Einträge gefunden werden sollen, nicht unbedingt genau so eingegeben werden müssen, wie sie gespeichert sind. Zu diesem Zweck befinden sich am Schluß des Programms einige DATA-Zeilen, die abwechselnd eine Original-Zeichenfolge und eine enthalten, die statt dessen gespeichert wird. So wird PH zu F, Y zu I, AI zu EI, C zu K und so weiter. Die weiter hinten stehenden Umcodier-Vorschriften müssen dabei bereits die vorne stehenden berücksichtigen, z.B. KOMPUTER statt COMPUTER. Steht hinter dem Begriff am Zeilenende noch ein Komma bzw. stehen innerhalb einer DATA-Zeile zwei aufeinanderfolgende Kommas, so entspricht das einem „leeren" String, d.h. dieser Begriff wird ersatzlos eliminiert (z.B. UNG, ING, TEST usw.). Selbstverständlich ist es jederzeit möglich, diese Umcodier-Vorschriften für andere Themengebiete umzuformulieren. Damit man automatisch bei Themawechsel (Laden einer anderen Datei von Diskette) mit den korrekten Umcodier-Vorschriften arbeitet, werden diese stets mitgeladen.

Jeder Eintrag besteht aus einer später beim Auffinden auszugebenden Zeile (z.B. Titel eines Zeitschriften-Beitrags und die zugehörige Heft-Nummer) sowie aus den Suchbegriffen, unter denen er später auffindbar sein soll. Das Umcodieren der Suchbegriffe erfolgt automatisch bereits während der Eingabe und nicht jedesmal während des Suchvorgangs, was erheblich Zeit spart.

Nach RUN erscheint auf dem Bildschirm eine Liste der zur Verfügung stehenden Kommandos; mit H kann man diese zu jeder Zeit wieder sichtbar machen. Kommando E dient der Eingabe eines Eintrags sowie der zugehörigen Suchbegriffe (max. 5 pro Eintrag). Werden weniger als fünf Suchbegriffe benötigt, so drückt man statt des nächsten einfach nur die Return-Taste.

Mit K ist es möglich, eine Liste aller Einträge sowie der bereits umcodierten Suchbegriffe auszugeben. Das eigentliche Suchen erfolgt mit dem Kommando A.

```
100REM --- VARIABLEN ---
105REM A$(J,0)    AUSZUGEBENDE DATEN
110REM A$(J,1-5)  KOMPRIMIERTE SUCHWORTE
115REM B$         BEFEHLS-BUCHSTABEN
120REM D$         CHR$(4) FUER DOS
125REM E$(I,J)    ZU ERSETZENDE WORTE
130REM K$         TASTENDRUCK U.A.
135REM M$         TEILSTRING U.A.
140REM N$,N1$     FILENAME U.A.
145REM E          ZAHL DER EINTRAEGE
150REM F          FLAG F.DATEN GEFUNDEN
155REM I,J,K      LAUFVARIABLE
160REM SW         SUCHWORT-ZAEHLER
165REM Z          ZAHL DER ERSATZWORTE
170REM ZS         MAX.ZAHL DER SUCHBEGR.
175:
180REM --- INITIALISIERUNG ---
185ZS=5:DIMA$(400,ZS):PRINTCHR$(12);
190HOME:PRINT"MC-DATENBANK":GOSUB265
195B$="HLSEAXKD":D$=CHR$(4):DIME$(30,1)
200FORZ=0TO30:READN$:IFN$="END"THEN210
205E$(Z,0)=N$:READE$(Z,1):NEXT
210Z=Z-1
215REM --- KOMMANDOEINGABE ---
220PRINT:PRINT"KOMMANDO: ";
225GETK$:FORI=1TOLEN(B$)
230IFK$="Q"THENPRINT"<-- GOTO220":END
235IFK$=MID$(B$,I,1)THENPRINTK$:GOTO245
240NEXT:GOTO225
245IFI<4THENONIGOSUB265,385,435:GOTO220
250ONI-3GOSUB485,550,610,655,680
255GOTO220
260REM --- KOMMANDOLISTE ---
265PRINT:PRINT"H HILFE (DIESE LISTE)"
270PRINT"E EINGABE NEUER DATEN"
275PRINT"A SUCHEN UND AUSGEBEN"
280PRINT"X 1 EINTRAG  LOESCHEN"
285PRINT"L VON DISKETTE  LADEN"
290PRINT"S AUF DISK  SPEICHERN"
295PRINT"K KONTROLLE D.EINGABE"
300PRINT"D DOS-BEFEHL,Z.B.PR#1"
305PRINT"Q PROGRAMM  VERLASSEN":RETURN
310:
315N1$="":REM --- STICHW.CODIEREN ---
320FORI=1TOLEN(N$):M$=MID$(N$,I,1)
325IFM$>"Z"THENM$=CHR$(ASC(M$)-32)
330IFM$<"0"OR(M$>"9"ANDM$<"A")THEN340
335N1$=N1$+M$
340NEXTI:FORI=0TOZ:J=LEN(E$(I,0))
345FORK=1TOLEN(N1$)-J+1
350IFMID$(N1$,K,J)=E$(I,0)THEN360
355NEXTK,I:RETURN
360IFK=1THENK$=E$(I,1):GOTO370
```

Abb. 3.2.3.4 Datenbank-Programm in Applesoft-Basic mit schreibweisentoleranter Suche

```
365K$=LEFT$(N1$,K-1)+E$(I,1)
370N1$=K$+MID$(N1$,K+LEN(E$(I,0)))
375GOTO355
380REM --- VON DISK LADEN ---
385INPUT"NAME=";N$:IFN$<"A"THENRETURN
390PRINTD$"READ"N$:INPUTZ:IFZ<2THEN415
395FORI=0TOZ:INPUTE$(I,0),E$(I,1):NEXT
400INPUTZS,E:IFZS>50RZS<10RE<2THEN415
405FORI=0TOE-1:FORJ=0TOZS:INPUTA$(I,J)
410NEXTJ,I:PRINT"GELADEN":GOTO420
415PRINT"FALSCHE DATEI!"
420PRINTD$"CLOSE"N$:RETURN
425:
430REM --- AUF DISK SPEICHERN ---
435INPUT"NAME=";N$:IFN$<"A"THENRETURN
440IFE<2THENPRINT"KEINE DATEN":RETURN
445PRINTD$"OPEN"N$:PRINTD$"DELETE"N$
450PRINTD$"OPEN"N$:PRINTD$"WRITE"N$
455PRINTZ:FORI=0TOZ:PRINTE$(I,0)
460PRINTE$(I,1):NEXT:PRINTZS:PRINTE
465FORI=0TOE-1:FORJ=0TOZS:PRINTA$(I,J)
470NEXTJ,I:PRINTD$"CLOSE"N$:RETURN
475:
480REM --- EINGABE ---
485INPUT":";N$:IFN$<"0"THENRETURN
490A$(E,0)=N$:PRINT"SUCHWORTE ";
495PRINT"RETURN=ENDE":FORSW=1TOZS
500PRINT"NR."SW;:INPUT":";N$
505IFN$<"0"ANDSW=1THEN530
510IFN$<"0"THEN525
515GOSUB315:IFN1$=""THEN535
520A$(E,SW)=N1$:NEXTSW
525E=E+1:RETURN
530PRINT"MIN.1 SUCHBEGRIFF!":GOTO500
535PRINT"NICHT AUSREICHEND!":GOTO500
540:
545REM --- AUSGABE ---
550LF=0:REM LOESCH-FLAG
555INPUT"SUCHBEGRIFF: ";N$:F=0
560GOSUB315:IFN$<"0"THENRETURN
565IFN1$=""THENPRINT"GENAUER?":GOTO555
570N$=N1$:FORJ=0TOE-1:FORK=1TOZS
575I=LEN(N$):SW=LEN(A$(J,K))
580IFSW<ITHENI=SW:IFI=0THEN595
585I=LEFT$(N$,I)=LEFT$(A$(J,K),I)
590IFITHENF=1:PRINTA$(J,0):IFLFTHEN620
595NEXTK,J:IFF=0THENPRINT"NICHTS!"
600RETURN
605REM --- EINTRAG LOESCHEN ---
610IFE<1THENPRINT"KEINE DATEN":RETURN
615LF=1:GOTO555:REM EINTRAG SUCHEN
620PRINT"LOESCHEN:J/N? ";
625GETK$:IFK$="N"THENPRINTK$:GOTO595
```

zu Abb. 3.2.3.4

```
630IFK$<>"J"THEN630
635PRINTK$:E=E-1:FORI=0TOZS
640A$(J,I)=A$(E,I):A$(E,I)="":NEXT
645RETURN
650REM --- EINTRAEGE KONTROLLIEREN ---
655FORI=0TOE-1:PRINT:PRINT:PRINTA$(I,0)
660FORJ=1TOZS:PRINTA$(I,J);" ";:NEXT
665NEXTI:PRINT:RETURN
670:
675REM --- DOS-KOMMANDO ---
680INPUT"DOS-KOMMANDO: ";N$
685IFN$<"A"THENRETURN
690PRINTD$;N$:RETURN
695:
700REM --- SUCHBEGRIFFE UMCODIEREN ---
705DATA PH,F,Y,I,AI,EI,H,,CK,K,C,K
710DATA TZ,T,III,3,II,2,OU,O,UNG,,ING,
715DATA KOMPUTER,,PROGRAMME,,PROGRAMM,
720DATA APPLE2,APPLE,TEXTEDITOR,EDITOR
725DATA TEST,,VK64,K64,KOMMODORE64,K64
730DATA EWLETTPAKARD,P,FUER,,MIKRO,
735DATA TEXASINSTRUMENTS,TI,STRIK,BAR
740DATA EMBLER,,BILDSKIRMTEXT,BTX
745DATA END
```

zu Abb. 3.2.3.4

Bei Eingabe von X wird ebenfalls nach einem Suchbegriff gefragt, und jener Eintrag, in dem dieser Begriff vorkommt, kann gelöscht werden. Mit Q ist es möglich, das Programm abzubrechen.

Das Apple-Disketten-Betriebssystem DOS 3.3 wird noch in einem späteren Kapitel besprochen. Das Kommando D erlaubt die Eingabe von DOS-Befehlen wie z.B. CATALOG (Disk-Inhaltsverzeichnis), PR# 1 (Drucker in Slot 1 einschalten) usw. DOS-Kommandos wie OPEN, READ, WRITE und CLOSE werden auch in den Routinen zum Abspeichern und Laden der Einträge auf und von Diskette verwendet; dazu ebenfalls später mehr. Beim Umschreiben des Programms auf andere Computer sind es vor allem diese DOS-Befehle, die angepaßt werden müssen.

Tiere raten

Obwohl „Tiere raten" eigentlich ein Ratespiel ist, zeigt das Programm in *Abb. 3.2.3.5* wesentliche Aspekte sogenannter Expertensysteme. Darunter versteht man eine Hardware-Software-Konfiguration, die aus einem genau definierten Satz von Regeln durch Dialog mit dem Benutzer bestimmte Schlußfolgerungen ziehen kann. Das besondere an dem hier abgedruckten Programm ist jedoch, daß dieser Satz von Regeln nicht fest im Programm selbst steht, sondern dynamisch verändert werden kann: Das Programm lernt unter Anleitung des Benutzers hinzu! Der Themenkreis „Tiere raten" sei daher nur als ein gängiges Beispiel gedacht; genauso gut könnte man auch Regeln zur Diagnose von Krankheiten, zum Auffinden von Feh-

```
10DIMF$(300),N$(300,1):D$=CHR$(4):REM (C) FE/MC1984
20F$(0)="IST ES EIN SAEUGETIER?":E=0
30N$(0,0)="GOLDFISCH":N$(0,1)="SCHIMPANSE"
35REM ----- KOMMANDOEINGABE -----
40PRINT:PRINT"LADEN,SPEICHERN,ANZEIGEN,DIALOG,EDIT.?";
45GETJ$:FORI=1TO6:IFMID$("LSADEQ",I,1)=J$THEN60
50NEXT:GOTO45
60PRINTJ$:PRINT:ONIGOTO700,660,750,100,200,70
70END
90REM ----- DIALOGTEIL -----
100I=0:PRINT"ICH BIN BEREIT!":PRINT
110PRINTF$(I)" ";:GOSUB600:L=J
120IFN$(I,L)<"A"THENI=VAL(N$(I,L)):GOTO110
125PRINT"TRIFFT "N$(I,L)" ZU? ";:GOSUB600
130IFJ=1THENPRINT:PRINT"ES IST ALSO "N$(I,L)".":GOTO40
140REM NEUES OBJEKT EINTRAGEN
150PRINT"WELCHE J/N-FRAGE UNTERSCHEIDET ES"
160PRINT"VON "N$(I,L)"?":E=E+1:INPUT">";F$(E)
170PRINT"WIE LAUTET DIE ANTWORT (J/N)? ";
180GOSUB600:INPUT"WIE HEISST ES? ";N$(E,J)
190N$(E,1-J)=N$(I,L):N$(I,L)=STR$(E):GOTO40
195:
200REM ----- EINTRAEGE EDITIEREN -----
210INPUT"FRAGE NR.: ";I:GOSUB770
220INPUT"NEUER FRAGETEXT: ";J$:IFJ$>""THENF$(I)=J$
230INPUT"ANTWORT BEI 'N': ";J$:IFJ$>""THENN$(I,0)=J$
240INPUT"ANTWORT BEI 'J': ";J$:IFJ$>""THENN$(I,1)=J$
250IFI>ETHENE=I:REM ZEIGER AUF ENDE
260GOTO40
590:
600REM ----- JA/NEIN-ABFRAGE -----
610GETJ$:IFJ$<>"N"ANDJ$<>"J"THEN610
620J=0:IFJ$="J"THENJ=1
630PRINTJ$:RETURN
640:
650REM ----- AUF FLOPPY SPEICHERN -----
660INPUT"FILENAME:";N$:PRINTD$"OPEN"N$
670PRINTD$"DELETE"N$:PRINTD$"OPEN"N$:PRINTD$"WRITE"N$
680PRINTE:FORI=0TOE:PRINTN$(I,0):PRINTN$(I,1)
690PRINTF$(I):NEXT:GOTO730
695:
700REM ----- VON FLOPPY LADEN -----
710INPUT"FILENAME:";N$:PRINTD$"READ"N$:INPUTE
720FORI=0TOE:INPUTN$(I,0),N$(I,1),F$(I):NEXT
730PRINTD$"CLOSE"N$:GOTO40
740:
745REM ----- FRAGEN+ANTW.ANZEIGEN -----
750PRINT"ZAHL DER EINTRAEGE:"E
760FORI=0TOE:PRINT:PRINT"FRAGE "I": ";:GOSUB770:NEXT
765GOTO40:REM ANZEIGE EINES EINTRAGS:
770PRINTF$(I):L=LEN(N$(I,0)):IFL>17THENL=17
780PRINT"N:"N$(I,0);SPC(18-L);"J:"N$(I,1):RETURN
```

Abb. 3.2.3.5 Selbstlernendes Programm. Die Daten sind hier als Binärbaum organisiert (vgl. Abb. 3.3.4.6)

```
>RUN

LADEN,SPEICHERN,ANZEIGEN,DIALOG,EDIT.?D

ICH BIN BEREIT!

IST ES EIN SAEUGETIER? J
TRIFFT SCHIMPANSE ZU? N
WELCHE J/N-FRAGE UNTERSCHEIDET ES
VON SCHIMPANSE?
>IST ES SEHR GROSS?
WIE LAUTET DIE ANTWORT (J/N)? J
WIE HEISST ES? ELEFANT

LADEN,SPEICHERN,ANZEIGEN,DIALOG,EDIT.?D

ICH BIN BEREIT!

IST ES EIN SAEUGETIER? J
IST ES SEHR GROSS? J
TRIFFT ELEFANT ZU? J

ES IST ALSO ELEFANT.

LADEN,SPEICHERN,ANZEIGEN,DIALOG,EDIT.?Q
```

Abb. 3.2.3.6 Beispiel für den Dialog mit dem selbstlernenden Programm; Benutzereingaben sind unterstrichen

lern in einem bestimmten Computertyp oder zur Identifizierung von Virustypen realisieren.

Die Regeln sind hier in Form eines sogenannten „Binär-Baums" eingebaut. Das bedeutet, daß es zu einer vom Computer gestellten Frage genau zwei mögliche Antworten gibt, nämlich J und N. Jede dieser beiden Antwort-Möglichkeiten führt nun entweder zu einer weiteren Frage, die das Ziel noch genauer spezifiziert, oder (falls das Gewünschte schon gefunden wurde) zu einer eindeutigen Antwort.

Stellt der Benutzer nun aber fest, daß die ihm schließlich gelieferte Antwort nicht mit den Tatsachen der „richtigen Welt" übereinstimmt, so hat er die Möglichkeit, nach „TRIFFT XXXX ZU?" N für Nein einzugeben. Das Programm fragt ihn dann, welche Frage die eigentlich sinnvolle Antwort von der soeben gelieferten unterscheidet. Der Benutzer muß sich nun eine solche Frage überlegen und sie eingeben. Anschließend fragt ihn das Programm noch nach dem Namen des neuen Objekts (z.B. eines Tiers, das bisher noch nicht gespeichert war), der ebenfalls eingegeben werden muß. Ab sofort ist dann der Satz von Regeln um eine erweitert, und beim nächsten Mal ist das neue Tier kein unbekanntes Wesen mehr.

Abb. 3.2.3.6 zeigt ein Beispiel für den Dialog gleich nach RUN. Zunächst kennt das Programm nur die Tiere „Goldfisch" und „Schimpanse". Um zwischen beiden zu unterscheiden, fragt es nach dem Kommando D (Dialog) zunächst, ob es sich um ein Säugetier handelt. Der Benutzer dachte sich hier das Wort „Elefant", das

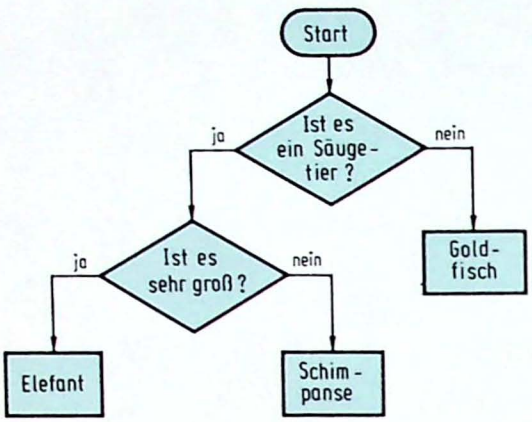


Abb. 3.2.3.7
So wird man zur gewünschten Antwort hingeführt

noch nicht implementiert war, und mußte es deshalb inklusive einer neuen Unterscheidungs-Frage neu eingeben (alle Benutzer-Eingaben sind hier unterstrichen abgedruckt). *Abb. 3.2.3.7* macht die Struktur des so entstandenen Binärbaums deutlich.

Damit das Programm beim Ausschalten des Rechners die neu gelernten Regeln nicht einfach vergißt, kann man diese mit dem Kommando S auf Diskette abspeichern und später mit L wieder laden. Schließlich hat man noch die Möglichkeit, sich den Binärbaum der Regeln auf dem Bildschirm anzusehen (Kommando A). Er ist hier in Form zweier Arrays organisiert: Das Feld F$(I) enthält die Fragen und N$(I,J) die zugehörigen Antworten. Die Elemente von N$ mit J=0 entsprechen der Antwort „Nein", die mit J=1 der Antwort „Ja". Steht in N$ eine Zahl, so bedeutet das, daß der „Baum" noch nicht zu Ende ist: Diese Zahl stellt den Index I der nächsten zu stellenden Frage dar. Enthält N$(I,J) dagegen einen String, der mit einem Buchstaben beginnt, so ist der Baum hier zu Ende, und das Programm stellt die Frage, ob N$(I,J) zutrifft.

Auch hier werden für das Abspeichern und Laden der Fragen und Antworten die typischen Apple-DOS-3.3-Befehle verwendet; sie müssen beim Umschreiben auf andere Computertypen bzw. Betriebssysteme angepaßt werden.

3.2.4 Forth

Forth ist eine etwas eigenwillige Programmiersprache, die vor allem bei maschinennahen Steuerungs-Anwendungen verbreitet ist. Sie wurde um 1968 von C. H. Moore entwickelt und später von E. H. Rather ergänzt, die beide am amerikanischen National Radio Oberservatory arbeiteten. Die erste Anwendung von Forth war deshalb auch die Steuerung großer Teleskope mit Mikrocomputern.

Forth unterstützt, ja erzwingt die saubere Strukturierung von Programmen: Es gibt keinen GOTO-Befehl wie in Basic. Man schreibt die Programme, indem man sich aus dem bisher vorhandenen Befehlssatz neue Befehle definiert. Auf diese Weise wird ein Gesamtproblem in mehrere kleinere Einzelprobleme aufgespalten, denen jeweils ein Name (ein neuer Forth-Befehl) zugeordnet wird. Das Hauptprogramm braucht dann nur noch die neu definierten Befehle in der richtigen Reihenfolge aufzurufen und erhält schließlich auch selbst einen Namen, mit dem es gestartet werden kann. Die Forth-Befehle nennt man auch „Verben" (Tätigkeitswörter), ihre Gesamtmenge „Dictionary" (Wörterbuch).

Forth wird gewöhnlich in Form eines Compilers implementiert und führt zu sehr schnellen Programmen mit wenig Overhead. Wenn ein sogenannter Target-Compiler zur Verfügung steht, ermöglicht er es sogar, den compilierten Objektcode z. B. auf einem kleinen Einplatinen-Computer laufen zu lassen.

Umgekehrte Polnische Notation

Im Gegensatz zu den meisten anderen Programmiersprachen werden arithmetische Ausdrücke nicht in algebraischer Form, sondern in der Umgekehrten Polnischen Notation (UPN; Reverse Polish Notation = RPN), da Parameter über den Stack übergeben werden. Dieser Stack ist wohlgemerkt nicht derselbe, der für die Speicherung von Unterprogramm-Rücksprungadressen verwendet wird; bei Forth gibt es einen Datenstack und einen Return-Stack. Wenn hier vom Stack die Rede ist, dann ist stets der Daten-Stack gemeint. Ein erstes Beispiel: Die Zahlen 5 und 7 sollen addiert werden. In Basic würde man das so schreiben:

```
PRINT 5+7
```

In Forth schreibt man dagegen:

```
5 7 + .
```

Der Punkt am Ende dieser Zeile bewirkt das Gleiche wie der Befehl PRINT in Basic. Auffallend ist, daß in Forth die Reihenfolge der Zahlen und Anweisungen völlig anders ist als in Basic. Auf dem Stack passiert bei diesen Anweisungen folgendes:

Stack-Register	vorher	5	7	+	.
X	—	5	7	12	—
Y	—	—	5	—	—
Z	—	—	—	—	—

X nennen wir hier (willkürlich) das oberste Stack-Element, Y und Z die jeweils nächsttieferen. (Natürlich passen auf den Stack mehr als nur drei Zahlen bzw. Elemente, aber für unsere Beispiele mögen drei genügen.) Vor der Eingabe einer Zahl sind die Stack-Register leer (—), ihr Inhalt ist nicht relevant. Die Eingabe der Zahl 5 legt diese im X-Register ab. Gibt man nun (durch ein Leerzeichen getrennt) eine weitere Zahl ein, hier die 7, so wird die 5 um eine Position weiter nach unten ge-

schoben. Der Forth-Befehl „+“ nimmt nun die beiden Werte im obersten (X) und nächsttieferen (Y) Stack-Register, addiert sie und legt das Ergebnis wieder in X ab. Der Befehl „.“ nimmt schließlich das oberste Element (hier das Ergebnis) vom Stack und gibt es auf den Bildschirm aus.

Besonders deutlich wird der Unterschied der UPN-Schreibweise gegenüber der algebraischen bei arithmetischen Ausdrücken mit Klammern, zum Beispiel:

$8-(3+2)=?$

Hier muß zuerst der Klammerausdruck berechnet werden, bevor er von 8 abgezogen werden kann. In Basic schreibt man dazu einfach: PRINT 8−(3+2). In Forth wird diese Aufgabe mit Hilfe des Stacks ganz ohne Klammern gelöst:

Stack	vorher	8	3	2	+	−	.
X	—	8	3	2	5	3	—
Y	—	—	8	3	8	—	—
Z	—	—	—	8	—	—	—

Das Forth-Programm benötigt hier, ohne die Reihenfolge der Zahlen selbst umzustellen, für das gleiche Problem also weniger Zeichen als ein gleichwertiges Basic-Programm, da die Klammern wegfallen. Allerdings erfordert die Umsetzung eines Klammer-Ausdrucks in UPN schon ein wenig Denkarbeit vom Programmierer.

Da Forth vom Konzept her vor allem für Steuerungsaufgaben geeignet ist, besitzen die meisten Forth-Compiler auch nur eine 16-Bit-Integer-Arithmetik (Wertebereich −32768...+32767). Fließkomma-Routinen sowie komfortable String-Verarbeitungs-Befehle muß man sich bei Bedarf selbst durch Neudefinition zusätzlicher Forth-Worte konstruieren. Forth ist daher weniger als Konkurrenz zu Basic oder Pascal zu sehen, sondern vielmehr als Alternative zur Assembler-Programmierung: Ein Forth-Programm ist durchschnittlich etwa zehnmal langsamer als ein optimiertes Maschinensprache-Programm und rund zehnmal schneller als ein interpretiertes Basic-Programm.

Forth-II

Für den Computer Apple-II und kompatible wurde von W. G. Graves ein Forth-Compiler entwickelt, der recht verbreitet ist. Er benötigt ein System mit min. 24 KByte Arbeitsspeicher sowie min. ein Disketten-Laufwerk. Folgende Verben stehen dem Benutzer zur Verfügung (beim Stack sind die Werte in der Reihenfolge der Ein- und Ausgabe angegeben, § = @):

Verb	Stack		Wirkung
	vorher	nachher	
((	— —	— —	Beginn einer rekursiven Befehlsfolge (Immediate Mode)
))	— —	— —	Ende der rekursiven Befehlsfolge
(	— —	— —	Beginn eines Kommentars im Quelltext

Verb	Stack vorher	Stack nachher	Wirkung
)	— —	— —	Ende des Kommentars
“	— —	n a	String folgt; die Adresse der Zeichenfolge a und seine Länge n kommen auf den Stack. Ein weiteres Anführungszeichen beendet den String
!	v a	— —	Wert v an die Adresse a speichern
§	a —	v —	Wert v von der Adresse a holen
‘	— —	a —	Adresse des Maschinenprogramms holen, das im Compiler das folgende Verb ausführt
*	a b —	c — —	Multiplikation (16 Bit): c=a*b
*/	a b c	d — —	Multiplikation mit 32-Bit-Ergebnis: d=a*b/c
/	a b —	c — —	Division: c=a/b
/MOD	a b —	r c —	Division: c=a/b, r=Rest
−	a b	c —	Subtraktion: c=a−b
+	a b	c +	Addition: c=a+b
+!	v a	— —	Zur Speicherzelle mit der Adresse a den Wert v addieren
+LOOP	v —	— —	v zum Index einer DO-LOOP-Schleife addieren; Befehle in der Schleife ggf. wiederholen
,	v —	— —	Wert v an der nächsten freien Position des Dictionary speichern
.	v —	— —	Wert v ausgeben
?	a —	— —	Inhalt der Speicherzelle a ausgeben
0=	v —	t —	Wenn v Null ist, wird t=1 (wahr), sonst t=0
:	— —	— —	Nachfolgende Verb-Definition compilieren
;	— —	— —	Ende einer Verb-Definition
;:	— —	— —	Compilieren beenden, Verb ausführen
<	a b	t —	t=1, wenn a<b, sonst t=0
>	a b	t —	t=1, wenn a>b, sonst t=0
<=	a b	t —	t=1, wenn a<=b, sonst t=0
>=	a b	t —	t=1, wenn a>=b, sonst t=0
<>	a b	t —	t=1, wenn a<>b, sonst t=0
=	a b	t —	t=1, wenn a=b, sonst t=0
<R	v —	— —	Wert v auf den Unterprogramm-Stack bringen
ABORT	— —	— —	Programm beenden
ADV	— —	— —	Ein Input-Zeichen überspringen
AND	a b	c —	Bitweise Verknüpfung von a und b ergibt c
AREG	— —	a —	CPU-Akku-Übergabe bei CALL
ASCB	n a	v 1	Der String an der Adresse a mit der Länge n
	oder	0 —	wird in einen numerischen Wert v umgewandelt. Wenn der String nichtnumerische Zeichen enthält, wird v=0

Verb	Stack vorher	Stack nachher	Wirkung
BASC	v n a	n a	Der Wert v wird in einen String ab der Adresse a mit der Länge n umgewandelt. Die Operation ignoriert zunächst n
BASE	— —	a —	Zahlensystem bei I/O-Operationen, z.B. 2 BASE !
BEGIN	— —	— —	Markiert den Anfang einer BEGIN-END-Schleife
BUFFER	— —	a —	Liefert den Anfang des Quelltext-Puffers
BUFTOP	— —	a —	Liefert die nächste freie Puffer-Adresse
C§	a —	v —	Byte von der Adresse a holen; die höherwertigen 8 Bits von v sind Null
C!	v a	— —	Niederwertigen Teil von v an die Adresse a speichern (ein Byte)
C.	c —	— —	ASCII-Zeichen mit Code c ausgeben
C,	c —	— —	Zeichencode c in das nächste Dictionary-Byte speichern
CALL	a —	— —	Sprung zum Maschinenprogramm an der Adresse a
CHARS	— —	a —	Adresse einer Tabelle mit den ASCII-Zeichen 0123456789ABCDEF (Hex-Ziffern)
CLR	— —	— —	Textpuffer löschen
CONSTANT	v —	— —	Konstante mit dem Wert v definieren, z.B. 10 CONSTANT ZEHN
CR	— —	— —	Return-Zeichen ausgeben
DEC	— —	— —	BASE auf 10 setzen (dezimale Ein-/Ausgabe)
DEFERR	— —	— —	Gibt das letzte beim Compilieren gelesene Wort und UNDEFINED aus
DICT	— —	a —	Liefert die Adresse des letzten Verbs im Dictionary
DISPLAY	— —	— —	Zeigt den Textpuffer seitenweise an (Stop = CTRL-C, weiter = Return)
DO	e i	— —	Beginn einer DO-LOOP-Schleife mit Index-Anfangswert = i und Endwert = e; i und e werden auf dem Return-Stack gespeichert
DOS:	— —	— —	DOS-Kommando ausführen, z.B. DOS: CATALOG
DROP	a —	— —	Oberstes Stack-Element beseitigen
DTOP	— —	a —	Adresse des nächsten freien Bytes im Dictionary
DUP	a —	a a	Dupliziert das oberste Stack-Element
ELSE	— —	— —	Die folgenden Befehle werden ausgeführt, wenn die Bedingung nach IF nicht erfüllt ist

Verb	Stack vorher	Stack nachher	Wirkung
END	a —	— —	Endmarkierung einer BEGIN-END-Schleife. Die Schleife wird wiederholt, wenn a Null ist
EOP?	— —	— —	Falls Bildschirm-Ende erreicht, auf Tastendruck warten
EXEC	a —	— —	Ein Forth-Verb wird ausgeführt, dessen Adresse a auf dem Stack ist
FLUSH	— —	— —	Retten der Zeiger DICT und DTOP, so daß die Neueinträge auch bei späterem Neustart von Forth zur Verfügung stehen
FORGET	— —	— —	Nachfolgendes Verb löschen. (Vorsicht: Damit werden auch alle Verben gelöscht, die später als das angegebene Verb definiert wurden!)
GETVERB	— —	f a 1	Nachfolgendes Verb im Dictionary suchen;
	oder:	0 — —	f = Adresse, a = Adr. des folgenden Verbs. Wenn das Verb nicht gefunden wurde, wird nur eine Null auf den Stack geschrieben
HEX	— —	— —	Setzt BASE auf 16 (hexadezimale Ein-/Ausgabe)
HT	— —	— —	Momentane Cursorposition, z.B. HT C§
I	— —	v —	DO-Index vom Return-Stack auf den Daten-Stack kopieren; liefert den Wert der innersten DO-Index-Variable
IF	t —	— —	Die Befehle nach IF werden nur ausgeführt, wenn t nicht Null ist
IMME-DIATE	— —	— —	Setzt Immediate-Modus: Selbst wenn ein Verb nur innerhalb einer Definition genannt wird, wird es sofort ausgeführt
IN #	s —	— —	Eingabe auf Slot s umschalten
INPUT	— —	a —	Adresse des nächsten Zeichens im Eingabepuffer
INTOP	— —	a —	Adresse des letzten Zeichens im Eingabepuffer
J	— —	v —	Liefert den Wert der zweitinnersten DO-Index-Variable
LEAVE	— —	— —	DO-Index auf den Endwert setzen
LIN	— —	— —	Liest eine Zeile von der Tastatur, Anfangs- und Endadresse in INPUT und INTOP
LIST	— —	— —	Alle monetan verfügbaren Verben listen
LOAD	— —	— —	Text im Puffer compilieren u. ggf. ausführen

Verb	Stack vorher	Stack nachher	Wirkung
LOOP	— —	— —	Ende einer DO-Schleife. Der Index wird um 1 erhöht und die Schleife wiederholt, bis der Endwert erreicht ist
MAIN	— —	— —	So heißt das Forth-Programm selbst
MAIN2	— —	— —	Hauptschleife des Forth-Systems, ermöglicht rekursive Strukturen, da die momentane Eingabezeile weiter verarbeitet wird. MAIN2 muß ein Verb wie NAME R> DROP ; aufrufen
MINUS	a —	b —	Vorzeichenwechsel: b = − a
MOVE	n a b	— —	n Bytes ab Adresse a nach Adresse b kopieren; das Byte an Adresse a wird zuerst kopiert
MOVEUP	n a b	— —	Wie MOVE, jedoch wird das Byte an der Adresse a + n − 1 zuerst kopiert
NAME	a —	— —	Wort an der Adresse DTOP mit Sprungbefehl zur Adresse a ins Dictionary eintragen
NSTERR	— —	— —	Wort an der Adresse DTOP und NESTING ERROR ausgeben
NUMBER	— —	v 1	Wie ASCB, jedoch wird die Adresse des Strings nicht vom Stack, sondern von DTOP geholt
OR	a b	c —	Oder-Verknüpfung: c = a OR b
OVER	a b —	a b a	Zweites Stack-Element über das erste kopieren
PAGE	— —	— —	Bildschirm löschen
R>	a —	— —	a auf den Return-Stack bringen
RCH	— —	c —	Nächstes Zeichen von der Eingabezeile holen, INPUT um 1 erhöhen, ggf. neue Zeile holen
ROT	a b c	b c a	Die obersten drei Stack-Elemente tauschen
S.	n a	— —	String der Länge n ab der Adresse a ausgeben
S=	n a b	t —	n Zeichen der Strings ab den Adressen a und b vergleichen; t = 1, wenn gleich, sonst t = 0
SKIP	— —	— —	Im Eingabe-String bis zum ersten Zeichen vorrücken, das weder ein Leerraum noch ein Return-Zeichen ist
SPACES	n —	— —	n Leerräume ausgeben
SWAP	a b	b a	Vertauscht die beiden obersten Stack-Elemente

Verb	Stack vorher	Stack nachher	Wirkung
SYSERR	n a	— —	Von WORD gespeicherten Namen am Dictionary-Beginn sowie den String an der Adresse a und der Länge n ausgeben
THEN	— —	— —	Ende einer IF-ELSE-THEN- oder IF-THEN-Struktur
TOGGLE	n p	— —	Ton erzeugen; n = Zahl der Perioden, p = Periodendauer
U<	a b	t —	Vergleich zweier vorzeichenloser Zahlen (0...65535); t=1, wenn a<b, sonst t=0
U>	a b	t —	vgl. U<
U<=	a b	t —	vgl. U<
U>=	a b	t —	vgl. U<
U.	a —	— —	Vorzeichenlose Zahl a ausgeben
UBASC	v n a	n a	Vorzeichenlosen Wert v in einen String ab der Adresse a umwandeln. Dessen Länge n wird bei zunächst ignoriert
VARIABLE	v —	— —	Variable mit Anfangswert v definieren; Beispiel: 43 VARIABLE ANZAHL
VT	— —	a —	Liefert die Nummer der Zeile, in der sich der Cursor gerade befindet
WORD	c —	— —	Eingabestring bis zum Zeichen mit dem Code c ins Dictionary aufnehmen; DTOP erhält die Länge, der String beginnt zwei Bytes später. c selbst wird nicht mitgespeichert
XREG	— —	— —	Variable zur Übergabe des CPU-X-Registers bei CALL
YREG	— —	— —	Variable zur Übergabe des CPU-Y-Registers bei CALL

Mit dem Graves-Forth wird ein Editor zum Erstellen von Forth-Quelltexten mitgeliefert, der selbst in Forth geschrieben ist und mit LOAD EDITOR compiliert werden kann. Er besitzt folgende zusätzlichen Befehle:

ADD	— —	— —	Neue Zeilen ans Ende des Textpuffers anhängen
CH	— —	— —	Zeichenfolge überall im Quelltext austauschen, z. B. CH 1 40 ALT (Return) TO? NEU zum Ersetzen von ALT durch NEU in den Quelltext-Zeilen 1 bis 40
DEL	— —	— —	Löschen von Zeilen, z. B. DEL 2 oder DEL 2 5

Verb	Stack vorher	Stack nachher	Wirkung
ED	— —	— —	Textzeile editieren, z.B. ED 5
EDITOR	— —	— —	Name des Editors, z.B. FORGET EDITOR
FIND	— —	— —	Auftreten einer Zeichenfolge in einem Zeilenbereich suchen, z.B. FIND 5 12 TEXT
INS	— —	— —	Vor einer Zeilennummer neue Zeile(n) einfügen, z.B. INS 5
L	— —	— —	Zeilen listen, z.B. L 5 oder L 5 10

Will man in Forth-II ein kleines Programm zur Konversion dezimaler in hexadezimale Zahlen schreiben, so ist das extrem einfach, weil man mit BASE das Zahlensystem für Ein- und Ausgaben umschalten kann. Das Programm soll mit dem Namen CONVERT aufgerufen werden können, also muß man folgende Verb-Definition schreiben:

```
: CONVERT HEX . DEC ;
```

Das ist das komplette Forth-Programm! CONVERT ist sein (willkürlich gewählter) Name. Es wird angenommen, daß sich das System vor dem ersten Aufruf im Dezimal-Modus befindet. HEX schaltet vor dem Ausgabe-Befehl „." auf das hexadezimale System um und DEC anschließend auf dezimale Ein-/Ausgabe zurück. Will man nun die Dezimalzahl 16 in das hexadezimale System umwandeln, so gibt man nur ein:

```
16 CONVERT
```

Und schon erscheint die Hex-Zahl 10 auf dem Bildschirm.

FIG-Forth

Die weitaus größte Verbreitung hat der von der Forth Interest Group (Forth-Interessen-Gruppe) propagierte Forth-Dialekt erlangt — vor allem, weil die FIG für praktisch alle gängigen Mikroprozessortypen Forth-Compiler gegen einen geringen Unkostenbeitrag abgibt (FIG, P.O.B. 1105, San Carlos, CA 94070, USA).

Es wäre hier mehr verwirrend als zweckmäßig, alle theoretisch im FIG-Forth verfügbaren Befehle aufzulisten, da ein erheblicher Teil davon in erster Linie vom Compiler selbst für systeminterne Aufgaben verwendet wird. Die folgenden Befehle entsprechen in ihrer Wirkung dem soeben besprochenen Apple-Forth:

!	'	(	)	*
*/	+	+!	+LOOP	–
.	/	/MOD	:	;
<	=	>	<R	?
§	ABORT	AND	BASE	BEGIN
BUFFER	C§	C!	C?	CONSTANT
CR	DO	DROP	DUP	EDITOR
ELSE	END	FLUSH	FORGET	HEX
I	IF	IMMEDIATE	LEAVE	LOAD
LOOP	MINUS	MOVE	NUMBER	OR
OVER	R>	ROT	SPACES	SWAP
THEN	TOGGLE	U<	U.	VARIABLE
WORD				

Ferner gibt es Entsprechungen von Worten, die sich nur geringfügig in der Schreibweise unterscheiden:

Apple-Forth	FIG-Forth
EXECUTE	EXEC
DEC	DECIMAL
MAIN	FORTH
MAIN2	INTERPRET
THEN	ENDIF

Bei den übrigen Befehlen seien hier nur die wichtigsten aufgeführt. Wie beim Apple-Forth werden die vorwiegend systemintern verwendeten Worte hier nicht genannt (§ = @).

Verb	Stack vorher	Stack nachher	Wirkung
<#	—	—	Leitet Zahlen-Ausgabe als String ein
#>	v	a b	Beendet Zahlenumwandlung; liefert b Ziffern als String ab Adresse a ab
#	b —	b —	Wandelt eine Stelle einer 32-Bit-Zahl a je nach BASE in eine Ziffer b
#S	a —	b —	Wandelt je nach BASE a in Ziffernstring um
–DUP	v —	v v —	v wird dupliziert, falls v>0
	oder	v	
0<	v —	t —	t = 1, wenn v>0, sonst t = 0
0=	v —	t —	t = 1, wenn v=0, sonst t = 0 (=NOT)
?TERMINAL	—	t —	t = 1, wenn Taste gedrückt
ALLOT	a —	—	Reserviert a ungenutzte Bytes im Dictionary
B/BUF	—	v —	Liefert Länge eines Disk-Blocks

Verb	Stack vorher	Stack nachher	Wirkung
BLANKS	a b —	—	b Bytes ab Adresse a mit hex 20 füllen
BLK	—	v —	Liefert Nummer des aktuellen Disk-Blocks
BLOCK	a —	b —	Block a von Disk ab Adresse b lesen
CONTEXT	—	a —	Liefert die Adresse a eines Zeigers zum zuerst durchsuchten CONTEXT-Vokakular
COUNT	a —	b c —	Wandelt String, der mit einem Längenbyte beginnt, in die für TYPE nötige Form
CURRENT	—	a —	Liefert die Adresse eines Zeigers zum aktuellen Vokabular
D+	a b —	c —	32-Bit-Addition; c = a + b
D.	v —	—	Gibt 32-Bit-Zahl aus
D.R	v —	—	Gibt 32-Bit-Zahl rechtsbündig aus
DABS	a —	b —	Bildet den Absolutwert einer 32-Bit-Zahl
DMINUS	a —	b —	Vorzeichenwechsel einer 32-Bit-Zahl
EMIT	c —	—	Gibt Zeichen mit Code c aus
EMPTY-BUFFERS	—	—	Löscht alle Puffer
ERASE	a b —	—	b Bytes ab Adresse a mit Null füllen
EXPECT	a b —	—	b Bytes ab Adresse a von der Tastatur holen
FILL	a b c	—	b Bytes ab Adresse a mit c füllen
FLUSH	—	—	Schreibt alle geänderten Puffer (vgl. UPDATE) auf Diskette
HERE	—	a	Liefert die nächste freie Dictionary-Adresse
IN	—	v	Liefert Eingabe-Puffer-Offset für WORD
KEY	—	c —	Wartet auf Tastendruck, c = Zeichencode
LIST	a —	—	Ausgeben eines Bildschirms mit der Ordnungszahl a von Disk (vgl. Apple-LIST!)
+LOOP	v —	—	wie LOOP, jedoch mit Inkrement = v
M/MOD	a b —	c d —	Vorzeichenlose 32-Bit-Zahl a durch b dividieren, c = 16-Bit-Rest, d = 32-Bit-Quotient
MOD	a b —	c —	c ist der Rest der Division a/b
MIN	a b —	c —	c ist die kleinere der Zahlen a und b
MAX	a b —	c —	c ist die größere der Zahlen a und b
R	—	r —	r wird vom Return-Stack auf den Datenstack kopiert

Verb	Stack vorher	Stack nachher	Wirkung
.R	v	—	Gibt v rechtsbündig aus
REPEAT	—	—	Sprung zum letzten BEGIN
SCR	—	v	Liefert Nummer des von Disk geladenen Bildschirms (Screen)
SIGN	a b —	c —	Fügt Vorzeichen von a in den Ziffernstring b ein
SMUDGE	—	—	Invertiert das Gültigkeits-Flag-Bit des zuletzt definierten Worts. Solange dieses noch 1 ist, kann das Wort nicht aufgerufen werden. SMUDGE wird mit „;" automatisch aufgerufen, kann aber z.B. für rekursive Programme auch vom Programm benutzt werden
SP§	—	a —	Liefert aktuelle Stackpointer-Position
SPACE	—	—	Gibt einen Leerraum aus
TOGGLE	a b —	—	Speicherzelle a mit Maske b EOR-verknüpfen
TYPE	a b —	—	b Bytes ab Adresse a als String ausgeben
UPDATE	—	—	Markiert benutzten Puffer als geändert
VLIST	—	—	Gibt alle Verben des CONTEXT-Vokabulars aus
VOCABULARY name	—	—	Eröffnet ein neues Vokabular
WHILE	t —	—	Befehle nur ausführen, wenn t = 1 (z.B. in BEGIN/WHILE/REPEAT)

Wie Apple-Forth ist FIG-Forth weder ein reiner Interpreter (der die Forth-Worte bzw. Tokens interpretieren würde) noch ein Objektcode erzeugender Compiler. Vielmehr handelt es sich um ein Zwischending, das man als „Compreter" bezeichnen könnte: Das compilierte Forth-Programm ist einfach eine Liste von Adressen der den Worten zugehörigen Unterprogramme im Forth-Kern. Dies ist im Vergleich zur unmittelbaren Objektcode-Erzeugung einfacher zu handhaben, andererseits aber kaum langsamer. Von der Geschwindigkeit her ist es daher gerechter, von einem Compiler zu sprechen.

Rekursive Programme

Mehrmals tauchte in diesem Kapitel das Wort „rekursiv" auf. Ein Beispiel soll klären, was damit gemeint ist: Zu errechnen sei die Fakultät der Zahl 7, was in der Algebra auch als 7! geschrieben wird. Die Fakultät errechnet man durch Multiplika-

tion aller natürlichen Zahlen ab 1 bis einschließlich der Zahl, deren Fakultät gebildet werden soll, also hier:

7! = 1*2*3*4*5*6*7 = 5040 oder allgemein:
n! = 1*2*...*n

Zusätzlich gilt: 1! = 1, 0! = 1. Um ein Programm zu schreiben, das die Fakultät für beliebige Werte ermittelt, gibt es mehrere Lösungsansätze. Einer ist, die Multiplikation in einer DO-Schleife vom Anfangswert 1 (oder 2, da eine Multiplikation mit 1 ja nichts verändert) bis zum Endwert (hier n = 7) zu wiederholen.

Bei Forth liegt ein zweiter Weg nahe, der sich aus der Tatsache ergibt, daß ein Problem unter Forth einfach als Definition eines neuen Verbs geschrieben wird. Die Fakultät einer Zahl n könnte man auch so schreiben:

n! = n * (n − 1)!

Im konkreten Fall n = 7 würde man dann folgende Rechenschritte anwenden:

7! = 7 * 6!
6! = 6 * 5!
usw. bis
1! = 1

Natürlich muß sich der Computer bei jedem Rechenschritt merken, was eigentlich ursprünglich errechnet werden sollte. Im obigen Fall müßte er sich zunächst die oberste Zeile merken, während er 6! decodiert, die zwei obersten Zeilen, während er 5! decodiert und so weiter. Hierzu wird der Forth-Return-Stack (Unterprogramm-Stack) verwendet, indem auf ihn die Adresse des rekursiven Worts gelegt wird. Ist das Programm bei der letzten Zeile angelangt, stehen also sechs Adressen auf dem Stack. Dann arbeitet es sich von unten nach oben zurück, wobei nacheinander die Fakultäten von 2...7 entstehen.

Die Implementationen rekursiver Techniken sehen bei den einzelnen Forth-Versionen leider recht unterschiedlich aus. Bei FIG-Forth benötigt man die beiden Zeichen Ä und Ü (in ASCII entsprechen die Codes eckigen Klammern!) sowie das Wort SMUDGE. Damit wird ein noch nicht mit dem Zeichen „;“ fertig vereinbartes Verb sozusagen gewaltsam als fertig gekennzeichnet, kann also innerhalb einer Definition zwischen „:“ und „;“ rekursiv aufgerufen werden. Wichtig ist hierbei vor allem, daß mit Hilfe einer IF-Anweisung dafür eine Beendigung der Rekursions-Schleife gesorgt wird, sobald der Endwert erreicht wird, bei WORT als neu definiertem Verb also mit der Anweisungsfolge:

```
: WORT ...Anweisungen, Test auf Endwert
  IF Ä SMUDGE Ü WORT ENDIF ;
```

Rekursion ist übrigens keineswegs auf Forth beschränkt. Rein algorithmisch lassen sich Probleme auch in Pascal und Basic rekursiv lösen.

3.2.5 Pascal

Die Programmiersprache Pascal wurde nach dem französischen Mathematiker Blaise Pascal benannt und 1971 an der Eidgenössischen Technischen Hochschule Zürich (ETHZ) von Niklaus Wirth entwickelt, und zwar auf einem CDC-Großrechner. Heute gibt es zahlreiche Implementationen dieser Sprache auch auf Mikrocomputern, die vorwiegend im sogenannten P-Code geschrieben sind.

Der P-Code ist die Maschinensprache einer virtuellen CPU, deren Eigenschaften optimal an die Erfordernisse von Pascal angepaßt sind. Da es keinen Standard-Mikroprozessor gibt, der den P-Code direkt ausführen kann, ist ein Interpreter nötig, der die P-Code-Befehle auf einer herkömmlichen CPU simuliert.

Der Vorteil des P-Code-Konzepts ist, daß für die Implementierung des selbst in P-Code geschriebenen Pascal-Compilers auf unterschiedlichen CPU-Typen ausschließlich der P-Code-Interpreter umgeschrieben werden muß, der im Vergleich zum Pascal-Compiler selbst recht kurz ist. Nachteilig ist lediglich, daß die Interpretation des P-Codes natürlich Zeit kostet und deshalb die mit der CPU eigentlich erreichbare Leistungsfähigkeit nicht ausschöpft. Da eine Interpretation der Pascal-Worte selbst einen weiteren, unvertretbaren Geschwindigkeits-Verlust bringen würde, arbeiten fast alle Pascal-Implementationen als Compiler und nicht als Interpreter (Ausnahme: Macintosh-Pascal). Der Compiler liefert dabei den P-Code als Objektcode.

In der „University of California, San Diego" (UCSD) wurde ein auf dem P-Code basierender Pascal-Compiler entwickelt (UCSD-Pascal), der auch ein vollständiges Floppy-Disk-Betriebssystem einschließt und daher besonders bei Mikrocomputern weitere Verbreitung fand. Mit dem „Turbo-Pascal" des amerikanischen Herstellers Borland steht inzwischen aber auch ein Pascal-Compiler zur Verfügung, der nicht auf diesem P-Betriebssystem basiert, sondern sich in CP/M oder MS-DOS (siehe Kapitel über Betriebssysteme) einbinden läßt und eine erheblich höhere Compilier- und Arbeitsgeschwindigkeit als UCSD-Pascal bietet.

Jedes Programm in Pascal beginnt mit einem Kopf (PROGRAM) und endet mit einem Punkt. (Natürlich wird der Dezimalpunkt innerhalb einer Zahl nicht als Programmende gewertet!) Ein Programm besteht immer aus folgenden Teilen:

Programmteil	Beispiel
Programmkopf	PROGRAM UMFANG(INPUT,OUTPUT);
Deklarationsteil	CONST PI = 3.14159;
	VAR R,F : REAL;
Anweisungsteil	BEGIN
	READLN(R);
	F := 2*PI*R;
	WRITELN(F)
	END.

Der Übersichtlichkeit halber rückt man meist die zu einem bestimmten Programmteil gehörenden Teile um einige Leerzeichen ein. Auf die Wirkung des Programms selbst hat dies jedoch keinen Einfluß.

Im folgenden wird der Pascal-Grundwortschatz dargestellt, wie er an der ETH Zürich ursprünglich definiert wurde und auf dem nach wie vor alle Pascal-Implementationen aufbauen.

(* *) oder { }
Kommentartext, der beim Compilieren ignoriert wird. Beispiel:
(* DIES IST EIN KOMMENTAR *)

ABS
Funktion, die den Absolutwert einer Zahl liefert. Beispiel:
A := ABS(A);

ALFA
Nicht immer implementierter Variablentyp, mit dem zehn ASCII-Zeichen gepackt gespeichert werden können. Falls dieser Typ nicht zur Verfügung steht, kann er folgendermaßen simuliert werden:
TYPE ALFA = PACKED ARRAY [1...10] OF CHAR;

AND
Logischer Operator zum Verknüpfen zweier boolescher Werte (nicht bitweise!); liefert TRUE, wenn beide Werte TRUE sind.

ARCTAN
Funktion, die den Arcustangens einer im Bogenmaß angegebenen Zahl liefert. Beispiel:
A := ARCTAN(B);

ARRAY
Wort zum Definieren einer Array-Variablen. Beispiel:
ZAHL : ARRAY ['A'...'Z'] OF INTEGER;
definiert den Variablennamen ZAHL als ein Ganzzahl-Array, auf das mit den Buchstaben A bis Z als Index zugegriffen werden kann.

BEGIN
Definiert den Anfang eines Anweisungsblocks, der mit END beendet werden muß.

CASE
Führt je nach Variablenwert eine bestimmte Anweisung aus. Die Werteliste wird mit END abgeschlossen. Beispiel:

```
CASE TAG OF
 0 : WRITELN ('SONNTAG);
 1 : WRITELN ('MONTAG');
 2 : WRITELN ('DIENSTAG'); usw.
 ...
 6 : WRITELN ('SAMSTAG')
END;
```

CONST
Definiert eine Konstante. Beispiel:

```
CONST PI = 3.14159;
```

COS
Cosinus-Funktion, Argument muß im Bogenmaß vorliegen. Beispiel:

```
A := COS(B);
```

DIV
Operator für Ganzzahl-Division; im Unterschied zu „/" nötig, da hierbei u. U. keine ganze Zahl als Ergebnis erscheint. Beispiel:

```
Q := A DIV B    (* liefert Quotient als Integer-Zahl Q *);
R := A MOD B (* liefert zugehörigen Divisions-Rest *);
```

DISPOSE
Prozedur zum Löschen des mit NEW belegten Speicherplatzes für dynamisch zugewiesene Variablen, als Parameter dient eine Pointer-Variable. Beispiel:

```
DISPOSE (P, A, B);
```

DO
Steht vor einer Folge von Befehlen, die bei Erfüllung der nach WHILE genannten Bedingung oder so oft, wie in einer FOR-TO-Anweisung definiert, ausgeführt werden soll.

DOWNTO
Steht optional statt TO nach FOR, sorgt jedoch dafür, daß der Index nicht inkrementiert, sondern dekrementiert wird.

ELSE
Steht optional nach IF...THEN. Der folgende Befehl wird nur ausgeführt, wenn die IF-Bedingung nicht erfüllt ist. Falls mehr als ein Befehl nach ELSE stehen soll, ist die Folge mit BEGIN und END einzuschließen.

END
Beendet einen Programmteil (BEGIN, FUNCTION, PROCEDURE, CASE, RECORD) oder das gesamte Programm (PROGRAM). (Am Programmende muß zusätzlich ein Punkt stehen.)

EOF (F)
Diese Prozedur liefert den booleschen Wert TRUE, wenn die geöffnete Datei F (z. B. auf Diskette) zu Ende ist (End of FILE), sonst FALSE.

EOLN (F)
Liefert TRUE, wenn die Zeile in der Datei F zu Ende ist, sonst FALSE.

EXP
Liefert die e-Funktion einer Zahl. Beispiel:

```
A := EXP(B);
```

FILE
Definiert eine Datei. Beispiel:

```
DATEN : FILE OF REAL;
```

FOR
Beginn einer Wiederholungs-Anweisung. Im Gegensatz zu Basic wird kein NEXT verwendet; vielmehr werden die zu wiederholenden Befehle in eine BEGIN-END-Struktur eingeschlossen. Beispiel:

```
FOR I:= 0 TO 5 DO
  WRITELN(I);
```

FUNCTION
Definieren einer neuen Funktion in Pascal. Beispiel:

```
FUNCTION ARCSIN(X : REAL) : REAL;
  BEGIN
  ARCSIN := ARCTAN(X/SQRT(1-SQR(X)))
  END;
```

GOTO
Sprung zu einem Label (d.h. einer Sprungmarke im Programm). Labels sind positive ganze Zahlen mit maximal vier Stellen. Die Verwendung von GOTO ist in Pascal normalerweise nur bei „gewaltsamem" Verlassen eines Programmteils z.B. nach einem Fehler oder einer unzulässigen Eingabe sinnvoll und sonst eigentlich überflüssig. Beispiel:

```
GOTO 32;
```

GET(F)
Prozedur zum Übernehmen des nächsten Wertes aus der Disk-Datei F in den File-Puffer.

IF
Steht vor einer Bedingung. Der Befehl nach THEN wird dann nur ausgeführt, wenn die Bedingung erfüllt ist, der nach ELSE nur dann, wenn sie nicht erfüllt ist. (IF...THEN...ELSE gilt als nur eine Anweisung, deshalb darf vor ELSE kein Strichpunkt stehen!)

IN
Operator, der feststellt, ob ein Wert in einer bestimmten Menge enthalten ist. Beispiel:

```
READLN(STOFF);
IF STOFF IN ELEMENTE THEN WRITELN('HALBLEITER!');
```

LABEL
Definiert am Programmanfang (!) eine Sprungmarke für GOTO. Beispiel:

```
PROGRAM DEMO(OUTPUT);
LABEL 32;
BEGIN
  GOTO 32;
  WRITELN('NICHT ANZEIGEN');
32: WRITELN('HIER LABEL 32!')
END.
```

LN
Funktion; liefert den natürlichen Logarithmus (vgl. LOG in Basic!). Beispiel:

```
A := LN(B);
```

MAXINT
Pseudovariable, die den in der jeweiligen Pascal-Implementation größten möglichen Wert für Integer-Zahlen liefert (meist 32767).

MOD
Liefert den Rest einer Division. Beispiel:

```
A := B MOD C;
```

NEW
Standard-Prozedur in Pascal für Variablen vom Pointer-Typ (nicht immer implementiert!), dient zum Vergeben von Speicherplatz an dynamisch zugewiesene Variablen. Als Parameter dient ein Pointer (vgl. DISPOSE). Beispiel:

```
NEW(P,A,B);
```

NIL
Standardwert für nicht belegte Pointer in Pascal.

NOT
Operator zum Umkehren eines booleschen Wertes. NOT TRUE ergibt FALSE, NOT FALSE ergibt TRUE.

ODD
Diese Funktion liefert den Wahrheitswert TRUE, wenn das Argument eine ungerade Zahl ist. Beispiel:

```
FLAG := ODD(A);
```

OF
Wird in Zusammenhang mit der Deklaration von Variablen benötigt (z.B. bei CASE, FILE, SET und ARRAY).

OR
Logischer Operator; liefert TRUE, wenn mindestens einer der verknüpften Werte TRUE ist.

PACK
Macht aus einem „normalen" Feld ein PACKED ARRAY, um Speicherplatz zu sparen. Im folgenden Beispiel muß A als ungepacktes Feld definiert sein, B als das gepackte, I gibt an, bei welchem Element das Packen beginnen soll:

```
PACK(A,I,B);
```

PACKED
Wird bei Arrays verwendet, um ein Feldelement jeweils in einem Speicherwort (Byte) unterzubringen und so Speicherplatz zu sparen. Beispiel:

```
VAR A : PACKED ARRAY [1..9] OF CHAR;
```

PROCEDURE
Kennzeichnet den Beginn einer Prozedur, die mit einem Namen von anderen Programmteilen aufgerufen werden kann (ähnlich wie ein Unterprogramm in Basic). Ein erheblicher Unterschied ist die Möglichkeit, „lokale" Variablen zu definieren, deren Namen in anderen Programmteilen für ganz andere Zwecke verwendet werden bzw. sogar als anderer Typ definiert sein kann. Beispiel:

```
PROCEDURE TEST;
  VAR I : INTEGER;
  BEGIN ... END;
```

PROGRAM
Spezialfall von PROCEDURE: Das Gesamtprogramm wird nicht PROCEDURE, sondern PROGRAM genannt. Es muß nach dem zugehörigen END mit einem Punkt enden. Nach dem Namen wird angegeben, welche System-Prozeduren verwendet werden; normalerweise INPUT und OUTPUT für Ein- und Ausgabe. Beispiel:

```
PROGRAM TEST(OUTPUT);
```

PUT(F)
An die Datei F wird der Wert der Puffer-Variablen angefügt.

READ
Einlesen von Daten. Numerische Daten können voneinander durch nichtnumerische Zeichen (z.B. Leerräume) getrennt werden. Wenn die einzulesenden Variablen mit FILE als Disk-Datei deklariert wurden, erfolgt das Einlesen von Diskette. Die momentane Zeile wird von READ nicht beendet. Beispiel:

```
READ (A,B,C);
```

READLN
Wie READ, jedoch dann neue Zeile beginnen. Beispiel:

```
READLN(A,B,C);
```

RECORD
Deklaration eines Datensatzes. Im Gegensatz zu einem Array können die Elemente eines Records von beliebigem Typ sein, ihre Anzahl ist jedoch von vornherein festgelegt. Beispiel:

```
TYPE ZIFFER = '0'...'9'
     TELEFON = RECORD
                 VORWAHL: PACKED ARRAY [1..10] OF ZIFFER;
                 NUMMER  : PACKED ARRAY [1..10] OF ZIFFER;
                 END;
```

REPEAT
Wiederholungsanweisung. Die Befehlsfolge bzw. die Schleife zwischen REPEAT und UNTIL wird solange ausgeführt, bis die nach UNTIL stehende Bedingung erfüllt ist.

RESET(F)
Prozedur, die die Disk-Datei F an den Anfang zurückstellt und dem File-Puffer deren ersten Wert übergibt (falls die Datei nicht leer ist).

REWRITE(F)
Erzeugen einer leeren Disk-Datei namens F. Existierte die Datei vorher bereits, wird sie gelöscht.

ROUND
Rundungsfunktion. ROUND(A) entspricht TRUNC(A+0.5).

SET
Nicht bei allen Pascal-Compilern implementiert. Deklariert eine Menge von Elementen, die als Ganzes betrachtet wird; der Zugriff auf einzelne Elemente ist nicht möglich. Die Daten werden sehr platzsparend gespeichert. Beispiel:

```
TYPE BUCHST = SET OF 'A'..'Z';
```

SIN
Sinus-Funktion; liefert den Sinus einer im Bogenmaß angegebenen Zahl. Beispiel:

```
A := SIN(B);
```

SQR
Funktion, die das Quadrat einer Zahl liefert. Beispiel:

```
A := SQR(B);
```

TEXT
Datentyp für Text-Dateien.

THEN
Steht nach IF. Der Befehl bzw. die nach THEN zwischen BEGIN und END stehenden Befehle werden nur dann ausgeführt, wenn die Bedingung nach IF erfüllt (TRUE) ist.

TO
Gibt den Schleifen-Endwert in Verbindung mit FOR an. Beispiel:

```
FOR I:= 1 TO 5
```

TRUNC
Funktion, die den ganzzahligen Teil einer Zahl liefert. TRUNC(5.23) ergibt den Wert 5.

TYPE
Definiert die Werte, die ein neu „erfundener" Variablentyp annehmen kann und darf. Im Gegensatz z.B. zu Basic ist es so sehr einfach, unzulässige Werte im Programm mit einer Fehlermeldung abzufangen. Das Beispiel zeigt, wie man dafür sorgen kann, daß die beiden Variablen T1 und T2 Abkürzungen der Wochentage als Werte annehmen können:

```
TYPE TAG = (SO,MO,DI,MI,DN,FR,SA);
VAR T1,T2 : TAG;
```

UNPACK
Diese Standard-Prozedur wandelt ein PACKED ARRAY in ein „normales" Feld um, z.B. um bequemer darauf zugreifen zu können. Im folgenden Beispiel sei A als gepacktes und B als ungepacktes Feld definiert, während I das Element angibt, ab dem „ausgepackt" werden soll:

```
UNPACK(A,B,I);
```

UNTIL
Rücksprung zum letzten REPEAT, sofern die nach UNTIL stehende Bedingung erfüllt ist.

VAR
Beginn einer Variablen-Deklaration. Folgende Variablentypen sind möglich: BOOLEAN (Flag, kann TRUE oder FALSE als Wert annehmen); INTEGER (Ganzzahl-Variablen, meist −32768...+32767); REAL (Fließkomma-Zahlen, wie in Basic mit Exponent nach E), CHAR (einzelnes ASCII-Zeichen; Strings sind in Pascal nicht als Variablentypen vordefiniert!), FILE und RECORD (siehe dort). Im folgenden Beispiel werden die Variablen D und E dem Record C zugewiesen:

```
VAR A : INTEGER;
    B : REAL;
    C : RECORD
          D : INTEGER;
          E : BOOLEAN
        END;
```

WHILE
Wirkt im Zusammenhang mit DO. Der Befehl bzw. der mit BEGIN und END eingeschlossene Programmteil zwischen WHILE und DO wird ausgeführt, solange die Bedingung nach WHILE erfüllt ist. Beispiel:

```
WHILE I>1 DO I := I − 1;
```

WITH
Ermöglicht in einer nachfolgenden Anweisung die Verwendung von Komponenten-Namen einer Verbund-Variablen-Liste, ohne jeweils den Verbundnamen voranstellen zu müssen. Im folgenden Beispiel ist A die einem Record zugewiesene Verbundvariable, die B und C enthält:

```
VAR A : RECORD
          B : INTEGER;
          C : REAL
        END;
BEGIN
WITH A DO
  BEGIN
  B := 5;
  C := 12.65
  END;
END.
```

WRITE
Datenausgabe fortlaufend in einer Zeile, ohne diese mit Return zu beenden. Beispiel:

```
WRITE(A,B,C);
```

WRITELN
Datenausgabe wie WRITE, jedoch wird die Zeile dann mit Return beendet. Beispiel:

```
WRITELN('ERGEBNIS:',V);
```

Wenn ein Vergleich von Pascal mit Basic überhaupt zweckmäßig ist, so läßt sich objektiv folgendes feststellen: Ein wildes Hin- und Herspringen zwischen einzelnen Programmteilen, wie es in Basic mit GOTO und GOSUB möglich ist, wird in Pascal durch die Möglichkeit hinfällig, Prozeduren und Funktionen mit Namen zu versehen und sie, ähnlich wie in Forth, mit diesen Namen aufzurufen. Da man in Pascal ferner einzelnen Prozeduren eigene, lokale Variablen-Definitionen voranstellen kann, braucht man sich auch nicht wie in Basic bei jeder neu benötigten Variablen überlegen, ob sie eventuell an einer anderen Stelle des Programms benötigt wird und nicht verändert werden darf. Insgesamt ergeben sich in Pascal also fast automatisch sauber strukturierte, jederzeit nachvollziehbare Programme. Pascal eignet sich deshalb außer zum Programmieren auch gut zum system- und sprachenneutralen Formulieren algorithmischer Probleme.

Daß heute trotzdem häufiger in Basic als in Pascal programmiert wird, braucht man nicht unbedingt der Borniertheit von Basic-Freaks zuzuschreiben. Die meisten Basic-Implementationen arbeiten im Gegensatz zu den Pascal-Compilern als Interpreter (zumindest zunächst, ein späteres Compilieren des Interpreter-Programms ist ja nicht ausgeschlossen). Die Dialogfähigkeit eines Interpreters gegenüber der umständlichen Bedienung eines Compilers (Source schreiben, auf Disk

speichern, compilieren, Fehlermeldung, Source laden, wieder compilieren usw.) erleichtert die Programmentwicklung nicht nur dem Anfänger erheblich. Schließlich stehen z.B. in MBasic sehr komfortable Befehle für String-Operationen zur Verfügung, die es so in Pascal überhaupt nicht gibt und die man sich deshalb jedesmal erst mühsam als Pascal-Prozeduren schreiben müßte.

Insofern läßt sich also nicht sagen, daß eine der beiden Sprachen für alle denkbaren Anwendungen besser geeignet wäre als die andere, wenn auch manchmal über diese Frage unter den Gelehrten sehr heftig diskutiert wird.

UCSD-Pascal

Wie schon erwähnt, ist UCSD-Pascal bei Mikrocomputern recht verbreitet. Streng genommen müßte man es als Betriebssystem bezeichnen (UCSD-P-System), mit dem die Sprachen Pascal, Fortran-77, Compiler-Basic sowie Assembler für die CPU-Typen 8080, 8086, Z80, Z8, 6502, 6800, 6809, 68000 und 9900 realisiert wurden. Interessant ist auch die Möglichkeit, den von unterschiedlichen Quelltext-Sprachen (z.B. Pascal, Fortran und Basic) erzeugten P-Code später zusammenzubinden, so daß sich ein gemeinsames Objektprogramm ergibt. UCSD-P enthält einen Interpreter für den sogenannten P-Code, die Maschinensprache der virtuellen P-CPU, die ähnlich wie Forth stackorientiert arbeitet.

Die virtuelle P-CPU besitzt fünf Register: Programmzähler, Programm-Stackpointer (Return-Stack in Forth), Daten-Stackpointer (Daten-Stack bei Forth) sowie zwei Zeiger auf die Parameter-Deskriptoren und auf das Ende des Arbeitsspeichers. Der Compiler, der das Pascal-Programm in den P-Code übersetzt, ist selbst ein etwa 4000 Zeilen umfassendes Pascal-Programm (erhältlich bei Ch. Jacobi, ETHZ, Clausiusstr. 55, CH-8006 Zürich). Eine Benutzergruppe hat sich unter folgender Adresse etabliert: USUS, Chip Chain Secretary, P.O.B. 1148, La Jolla, CA 92038, USA.

Der Compiler arbeitet während des Übersetzungsvorgangs in sechs Schritten, wovon nur die letzten beiden maschinenspezifisch sind:

1. Decodieren der Pascal-Worte
2. Feststellen syntaktischer Fehler
3. Erzeugen der Tabellen für Labels, Variablen usw.
4. Feststellen von Ablauffehlern (Kontext-Fehler)
5. Zuordnen der Adressen für Variablen
6. Erzeugen des P-Codes

Das UCSD-System umfaßt folgende Einzelprogramme, die menügesteuert getrennt geladen werden müssen, so daß schon Computer mit 48 KByte Arbeitsspeicher ausreichen:

1. Bildschirm-Editor zum Erstellen des Quelltextes
2. Compiler zum Übersetzen der Pascal-Programme in den P-Code
3. Filer zur Durchführung von Disk-Datei-Manipulationen
4. Linker zum Verbinden getrennt compilierter P-Code-Programme
5. Library-Manager zum Verwalten einer Programm-Bibliothek
6. P-Code-Debugger (Monitorprogramm) zum Optimieren des erzeugten Objektcodes von Hand

Ein Objektprogramm darf länger sein, als es der Speicher eigentlich zuläßt: Das Betriebssystem sorgt durch SEGMENTs dafür, daß der jeweils nächste Teil als „Overlay" in den Speicher geladen und ausgeführt wird (ähnlich dem Befehl CHAIN in Basic). Neuere UCSD-P-Implementationen erlauben auch den quasiparallelen Ablauf mehrerer Programme (Multitasking).

Turbo-Pascal

Für Z80- und 8086/8088-Rechner gibt es seit 1983 einen recht leistungsfähigen und doch preiswerten Pascal-Compiler namens Turbo-Pascal von Borland, seit 1986 auch für die CPU 68000. Er entspricht bis auf wenige Details dem von N. Wirth definierten Standard: GET und PUT fehlen, GOTO ist nur innerhalb eines Programmblocks zulässig. Die Übergabe von Funktions- und Prozedur-Parametern ist nicht möglich. Die Definition von Konstanten, Variablen und Typen darf in beliebiger Reihenfolge erfolgen. Ferner gibt es einige nützliche Sprach-Erweiterungen.

Wesentliches Merkmal von Turbo-Pascal (der Name sagt es schon) ist die hohe Geschwindigkeit — sowohl beim Compilieren als auch bei der Programmausführung. Im Gegensatz zum UCSD-P-System wird wirklich Maschinencode für die reale CPU erzeugt, der sich mit dem Laufzeitpaket (Run Time Library) zu einem direkt ausführbaren Programm zusammenbinden läßt (.COM-Datei). Allerdings wird kein relokatibler Code erzeugt, so daß er sich nicht z.B. mit verschiebbaren Assembler-Objektdateien zusammenbinden läßt, was u.a. auch das Einbinden eines Hardware-Arithmetik-Prozessors erschwert. Relokatible Prozeduren lassen sich nur recht umständlich mit Pointer-Variablen und Heaps realisieren. Eine lokale Variable darf nicht mit VAR an ein Unterprogramm übergeben werden.

3.2.6 Weitere Programmiersprachen

Bei Mikrocomputern findet man, in der Reihenfolge der Häufigkeit, in erster Linie die Sprachen Basic, Assembler, Pascal und Forth. Andere Sprachen konnten sich

aus vielerlei Gründen weniger durchsetzen, werden aber oft von bestimmten Interessengruppen sehr propagiert. Schließlich ist jede Programmiersprache entstanden, indem die Entwickler einen Wortschatz geschaffen haben, der optimal an bestimmte Problemkreise angepaßt war — bei Forth waren dies ursprünglich astronomische Berechnungen. Es ist daher nicht möglich, zu behaupten, die eine oder andere Programmiersprache wäre grundsätzlich besser oder schlechter als irgendeine andere. Vergleiche sind nur unter Zugrundelegung der geplanten Anwendung sinnvoll. Im folgenden werden die Sprachen erwähnt, die sich ebenfalls einer gewissen Verbreitung bei Mikrocomputern erfreuen können. Exoten (und seien sie auch noch so sehr von bestimmten Gruppen favorisiert) werden absichtlich beiseite gelassen wie z.B. Algol und APL als typische Vertreter jener Sprachen, die nur auf Großrechnern und praktisch überhaupt nicht auf Mikros anzutreffen sind, oder Ada und Modula-2 als bislang wenig erfolgreiche Weiterentwicklung von Pascal.

Fortran

Die Sprache Basic ist ursprünglich aus dem um 1953 entstandenen Fortran-II entstanden. Fortran ist die Abkürzung von „Formula Translation", was auf den vor allem wissenschaftlichen Anwendungskreis hindeutet. Ähnlichkeiten von Basic und Fortran sind nicht zu übersehen: „Wildes" Umherspringen mit GOTO z.B. nach IF. Meist stehen wie in Basic zahlreiche mathematische Funktionen zur Verfügung, oft sogar mit doppelter Genauigkeit. Im Gegensatz zu Basic wird bei Ausgabebefehlen allerdings der Zeilenwechsel vor und nicht hinter eine Ausgabezeile angefügt. Der auf Mikrocomputern bekannteste Fortran-Compiler ist Fortran-80 von Microsoft, der sich an den ANSI-Standard Nr. 66 (Fortran IV) anlehnt. Dieser Compiler liefert einen sehr schnellen Objektcode, da z.B. Ausdrücke, deren Ergebnis konstant ist, schon beim Compilieren durch eine Konstante ersetzt werden (!). Allerdings stehen keinerlei String-Operationen zur Verfügung, so daß die Sprache praktisch nur für reine Rechenprobleme in Frage kommt.

PL/I

Um 1965 entstand bei IBM aus den Großrechner-Sprachen Algol und Fortran der als Universalsprache konzipierte „Hybrid" PL/I. Leider gab es zunächst sehr unterschiedliche Implementationen von PL/I. Erst als 1976 eine ANSI-Norm (Subset G) Klarheit schuf, entwickelte Digital Research den 2-Pass-Compiler PL/I-80 für die CPUs 8080/8085 und Z80.

Eine Besonderheit von PL/I ist die Möglichkeit, Variablen als binäre Fließkomma- (FLOAT) oder als BCD-Typen (DECIMAL) zu vereinbaren (auf die Problematik binärer Zahlendarstellung wurde bereits im Kapitel über Interpreter eingegangen). Ausgabe-Anweisungen erzeugen nie automatisch einen Zeilenwechsel; dieser muß stets mit SKIP erst erzwungen werden. Programmverzweigungen sind sowohl wie in Algol oder Pascal mit IF, THEN, ELSE bzw. REPEAT, UNTIL als auch wie in Fortran mit GOTO und Labels möglich. Im Gegensatz zu Pascal wer-

den unter PL/I DO-Schleifen schon vor der Abarbeitung auf die Endbedingung geprüft, so daß die Schleife nie „versehentlich" durchlaufen wird.

Cobol

Der Name Cobol entstand aus „Common Business-Oriented Language" — also eine kommerziell orientierte Sprache, die 1959 an Großrechnern entstand. Hauptmerkmal ist die Anlehnung an natürliche Sprache. Statt A = B/3 schreibt man in Cobol DIVIDE B BY 3 GIVING A. Dadurch werden auch einfache algebraische Formeln zu einem umfangreichen Programmstück, in dem man sich nachträglich kaum noch zurechtfindet: Cobol wurde für Kaufleute konzipiert, die keinerlei algebraisches Wissen mitbringen. Ähnlich wie beim UCSD-P-System wird vom Compiler üblicherweise kein von der CPU direkt ausführbarer Code erzeugt, sondern ein zu interpretierender Zwischencode, was zusammen mit der umständlichen Schreibweise zu relativ langsamen Programmen führt.

Auf der anderen Seite rechnet Cobol mit bis zu 18stelliger Genauigkeit im BCD-System, so daß keine Rundungs-Fehler zu befürchten sind. Die zur Verfügung stehenden Ausgabe-Operationen sind sehr komfortabel, was z.B. das Einhalten eines bestimmten Formular-Formats beim Drucker angeht. Diese Eigenschaften haben dafür gesorgt, daß Cobol heute nach wie vor im kommerziellen Bereich ab und zu verwendet wird. Die auf Mikrocomputern bisher meist verwendete Version ist Cobol-80 von Microsoft für die Prozessoren 8080, 8085 und Z80.

RPNL

Wie Forth basiert RPNL auf der umgekehrten Polnischen Notation (Reverse Polish Notation Language, z.B. von Lifeboat Software für CP/M-Rechner zu haben). Hauptvorteil ist der sehr geringe Speicherplatzbedarf von nur etwa 4 KByte für den Compiler, der auch während der Ausführung des generierten Objektcodes im Speicher stehen muß. Wie in Forth gibt es einen Daten- und einen Return-Stack, auch können neu definierte Namen (Verben) von später definierten aufgerufen werden. Die Verwendung rekursiver Befehlsfolgen ist in RPNL jedoch einfacher als in Forth und benötigt keine „Tricks". Das folgende RPNL-Programm berechnet rekursiv die Zweierpotenz einer Zahl:

```
COMPILE
PROGRAM ZWEIHOCH
  VAR HOCH    VAR WERT
  HOCH ? 0=   IF
  1 WERT :=
                ELSE
  HOCH ? DEC HOCH :=
  ZWEIHOCH
  WERT ? 2 *  WERT :=
                THEN
END
```

```
PROGRAM POTENZ
  WRITE 'ZWEI HOCH '
  HOCH READ
  ZWEIHOCH
  WRITE 'ERGIBT ' CR
  WERT PRINT
END
```

Ruft man das Programm durch Eingabe von POTENZ auf, so erscheint ZWEI HOCH als Eingabe-Aufforderung. Gibt man nun z.B. 10 ein, so schreibt die RPNL-Routine ERGIBT 1024 auf den Schirm.

C

Die Sprache C hat sich insbesondere in Zusammenhang mit dem Betriebssystem Unix (siehe dort) einen Namen gemacht, da Unix selbst in dieser Sprache geschrieben ist. Sie entstand aus der in den Bell Laboratories bei AT&T entwickelten Sprache B ursprünglich auf einem Minicomputer PDP-11 von DEC. Heute gibt es u.a. für die Mikroprozessoren 6502, 6809, 8086, Z8000 und 68000 C-Compiler. Zweck der Entwicklung war es in erster Linie, eine Sprache zu schaffen, die einerseits maschinennahe Probleme wie die Programmierung eines Betriebssystems relativ systemneutral (d.h. nicht in Assembler!) gestattet, die aber andererseits auch manche nützliche Eigenschaften höherer Sprachen realisiert.

Wie in Pascal müssen Variable als Typ deklariert werden (z.B. INT A,B;). Man kann jedoch den Compiler auch veranlassen, für zeitkritische Schleifen bestimmte Variablen in CPU-Registern zu halten. Inkrement und Dekrement z.B. der Index-Variablen I ist statt mit I=I+1 mit ++I und statt I=I−1 mit −− I schnell und effizient möglich. Für Verzweigungen und Schleifen gibt es in C Konstrukte wie IF...ELSE (ohne THEN!), CASE, FOR (ohne TO!) und WHILE...DO.

Wegen der teilweise sehr kurzen Schreibweise führt C zwar zu schnellen, aber leider auch zu nachträglich schwer lesbaren Programmen. Sie stellt somit vor allem eine Alternative zur Programmierung in Assembler dar und erlaubt es dabei, zu systemspezifische Software zu vermeiden. Ein erheblicher Teil kommerzieller System-Software wird heute in C geschrieben, so z.B. Betriebssysteme (Unix, Xenix, CP/M-68k u.a.) und Compiler für höhere Programmiersprachen.

Logo

Im Bereich schulischer Ausbildung spielt Logo eine gewisse Rolle; die Sprache wurde zunächst vor allem von Apple und Texas Instruments favorisiert, erlangte aber erst um 1982 eine größere Verbreitung, als Digital Research mit „Dr. Logo" eine portable Mikrocomputer-Implementation auf den Markt brachte.

Hauptmerkmal von Logo ist die Unterstützung der sogenannten Turtle-Grafik (Turtle = Schildkröte). Als Turtle bezeichnet man einen Grafik-Cursor, der mit einfachen Befehlen durch Angabe relativer Winkel und Schrittweiten in alle Rich-

tungen gedreht und bewegt werden kann. Auch einfache arithmetische Aufgaben lassen sich in Logo gut lösen, String-Operationen werden allerdings nicht unterstützt. Im Bereich kommerzieller Programme spielt Logo heute nach wie vor keine Rolle.

3.3 Grafische Darstellung von Programmen, Befehlen und Daten

3.3.1 Flußdiagramme

In welcher Reihenfolge ein Programm nacheinander was zu tun hat, wird überwiegend als Flußdiagramm oder Struktogramm veranschaulicht. Dies geschieht meist schon vor der eigentlichen Programm-Erstellung, d.h. in einer frühen Entwicklungsphase, zur anschaulichen Problemanalyse. Beide Methoden haben durchaus Vor- und Nachteile. Zunächst zum Flußdiagramm.

Ein Flußdiagramm gibt die einzelnen Schritte einer Problemlösung in ihrer zeitlichen Reihenfolge an, indem jeder Operation ein Kästchen zugeordnet wird *(Abb. 3.3.1.1)*. Alle Kästchen werden durch Pfeile so verbunden, wie es der tatsächlichen Ausführungs-Reihenfolge entspricht. Ovale Symbole kennzeichnen Beginn und Ende des Hauptprogramms oder einer Unterroutine, die mit einem Kästchen aufgerufen werden kann, das doppelte Seitenlinien enthält. Logische Entscheidungen werden mit Rauten symbolisiert, wobei es normalerweise die beiden „Ausgänge"

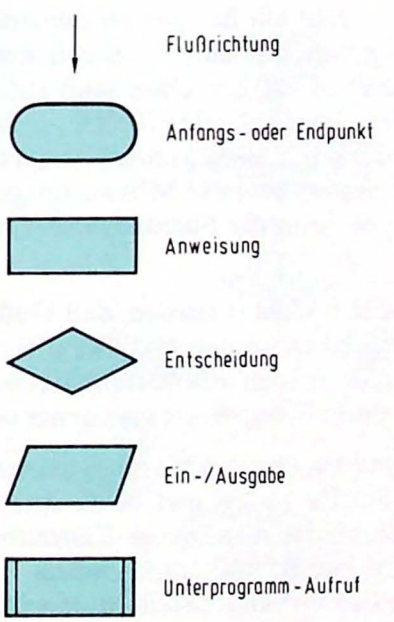


Abb. 3.3.1.1
Die wichtigsten Symbole für Flußdiagramme

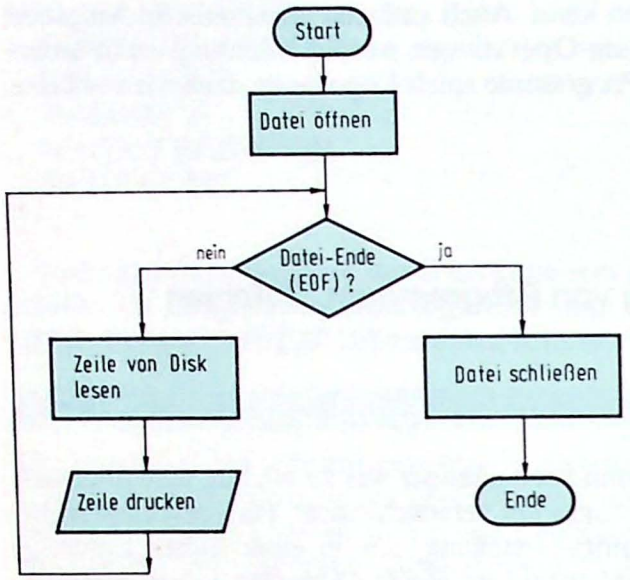


Abb. 3.3.1.2
Flußdiagramm eines Programms zum Ausdrucken einer Datei von Diskette

```
10 REM DATEI "ADR" AUSDRUCKEN
20 OPEN"I",1,"ADR"
30 IF EOF(1) THEN 50
40 INPUT#1,Z$:LPRINT Z$:GOTO 30
50 CLOSE 1:END
```

Abb. 3.3.1.3
So sieht das Programm in MBasic aus

Ja und Nein gibt. Kästchen mit schrägen Seitenlinien kennzeichnen Tastatur-Eingaben und Bildschirm-Ausgaben. (Manchmal werden auch weitere Symbole verwendet, deren Gebrauch ist jedoch uneinheitlich und eher verwirrend.)

Abb. 3.3.1.2 zeigt ein Beispiel für ein einfaches Flußdiagramm. Es veranschaulicht ein Programm, das eine Datei zeilenweise von Diskette lesen und auf einen Drucker ausgeben soll. Die Datei muß zunächst durch Angabe ihres Namens geöffnet werden. Sobald alle ihre Zeilen gelesen sind bzw. wenn sie ohnehin leer ist, muß sie wieder geschlossen werden. Andernfalls wird ihr Inhalt zeilenweise ausgedruckt. Wie man so etwas in MBasic programmiert, geht aus *Abb. 3.3.1.3* hervor. Die Rückverzweigung des Flußdiagramms entspricht hier dem Befehl GOTO 30 in Zeile 40.

Man kann sich leicht vorstellen, daß Flußdiagramme bei größeren Programmen sehr unübersichtlich werden und ihrem eigentlichen Zweck, nämlich den Ablauf des Programms verständlich zu machen, nicht mehr genügen. Um das zu vermeiden, gibt es zwei Lösungswege; man nennt sie „Top-Down" und „Bottom-Up".

Top-Down (von oben nach unten) bedeutet, daß man zunächst das Problem in sehr grobe Schritte zerlegt und sie zu einem entsprechend groben Flußdiagramm zerlegt, in dem jedes Anweisungs-Kästchen eine ganze Kette zusammengehöriger Anweisungen enthält oder auch einfach einen Unterprogramm-Aufruf darstellt. Wenn diese Grobstruktur erstellt ist, macht man sich daran, für jedes Teilproblem

bzw. Unterprogramm ein eigenes Flußdiagramm zu zeichnen und so das Problem immer mehr zu verfeinern, bis es schließlich bis ins kleinste Detail vollständig beschrieben ist.

Bottom-Up ist der genau umgekehrte Weg. Ausgehend von den bei der jeweiligen Programmiersprache überhaupt zur Verfügung stehenden Befehlen erstellt man Teildiagramme für jedes zu lösende Detailproblem. Das Hauptprogramm ist dann schnell formuliert bzw. als Flußdiagramm gezeichnet, indem man einfach nacheinander z.B. die Namen der Unterprogramme für die Teilprobleme hinschreibt.

Beide Methoden führen dazu, daß man oft geneigt ist, Programmstücke als Unterprogramme zu formulieren, obwohl der eigentliche Zweck von Unterprogrammen nur dann erfüllt wird, wenn die gleiche Befehlsfolge in einem Programm mehrmals vorkommt. Strukturiertes Programmieren, wie man diese systematischen Vorgehensweisen auch nennt, führt daher unter Umständen zu längeren, langsameren Programmen. Andererseits ist es nur so möglich, bei umfangreicher Software den Überblick nicht zu verlieren.

Flußdiagramme erzwingen strukturiertes Programmieren nicht. Mit den Richtungspfeilen kann man „wild" im Programm hin- und herspringen. Daß diese Möglichkeit unter Umständen durchaus auch sinnvoll sein kann, zeigt die Betrachtung des folgenden Problems.

Es sei wiederum eine Disk-Datei auf einen Drucker auszugeben. Allerdings soll das Drucken einer Datei sofort abgebrochen werden, wenn der Rechner feststellt, daß es sich um unsinnige Daten und somit um eine falsche Datei handelt. Beispielsweise wäre das der Fall, wenn eine Liste von Namen ausgedruckt werden soll, eine Zeile aber mit einer Ziffer beginnt. Dann soll der Rechner diese Datei wieder

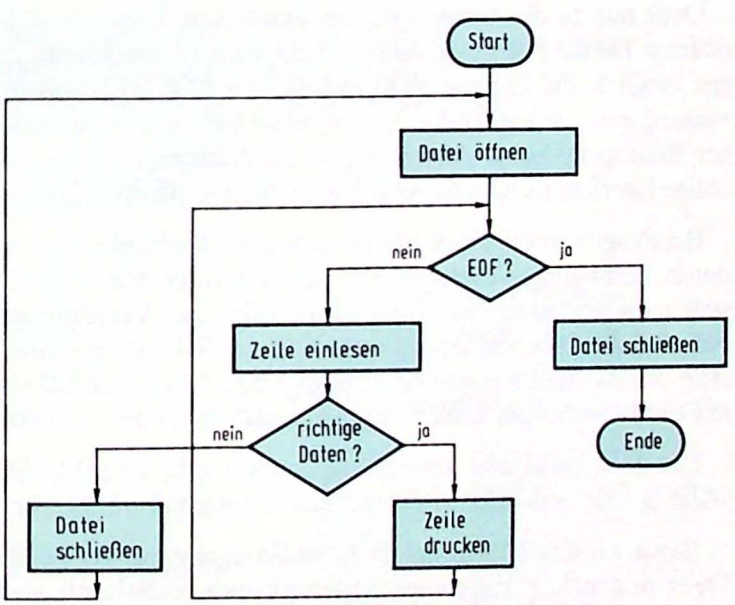


Abb. 3.3.1.4
Erweitertes Flußdiagramm mit Fehler-Abfrage

schließen und eine andere öffnen. Wie man diese Problemstellung in einem Flußdiagramm veranschaulichen kann, geht aus *Abb. 3.3.1.4* hervor. Schon die Tatsache, daß sich zwei Linien hier überkreuzen, zeigt die „Unstrukturiertheit" der Vorgehensweise, die übrigens typisch für die Fehlerbehandlung in Programmen ist. Denn bei Fehlern muß man meist den normalen Ablauf verlassen und sozusagen gewaltsam woanders hinspringen. (Die Struktur-Fanatiker selbst durchbrechen ihr heiliges Prinzip jedesmal bei Anwendung solcher Befehle wie ON ERROR GO-TO!) Mit dem nachfolgend beschriebenen Struktogramm, das sicher viele Vorteile bietet, lassen sich solche notgedrungen unstrukturierten Abläufe prinzipiell nicht ohne weiteres darstellen. Das Flußdiagramm ist hier also sozusagen unersetzbar.

3.3.2 Struktogramme

Wie schon gesagt, ist es (wenn man nicht von vornherein auf saubere Untergliederung achtet) bei Flußdiagrammen ohne weiteres möglich, wild zwischen den Anweisungs-Blöcken hin- und herzuspringen. Um dies zu verhindern, gibt es als Alternative die sogenannten Struktogramme. Sie heißen so, weil sie eine saubere Strukturierung erzwingen und beispielsweise eine falsche Schleifenbildung wie in *Abb. 3.3.2.1* bei FOR-NEXT-Anweisungen von vornherein gar nicht erst zulassen. Erlaubt ist nämlich (damit der Stack des Rechners nicht durcheinanderkommt) nur eine Verschachtelung wie in *Abb. 3.3.2.2;* die Flußlinien dürfen sich also nicht kreuzen!

Doch nun zu den Struktogramm-Symbolen. Eine einfache Ja-Nein-Bedingungs-Abfrage hat die Form von *Abb. 3.3.2.3;* es sind jedoch auch Mehrfach-Verzweigungen möglich, die in Basic dem Befehl ON I GOTO und in Pascal der CASE-Anweisung entsprechen *(Abb. 3.3.2.4).* Sind bei einer von mehreren Möglichkeiten einer Bedingung keine Anweisungen auszuführen, so kennzeichnet man das dann nötige Leerfeld durch eine sichtbar geringere Breite *(Abb. 3.3.2.5).*

Bei Programmschleifen gibt es mehrere Möglichkeiten. Man könnte sie natürlich durch Bedingungs-Abfragen am Anfang oder Ende realisieren. Struktogramme stellen für Schleifen aber eigene Konstrukte zur Verfügung. *Abb. 3.3.2.6* gilt für den Fall, daß die Abbruchbedingung jeweils vor Schleifenbeginn geprüft wird (WHILE). *Abb. 3.3.2.7* stellt die Abbruchbedingung an den Schluß (NEXT in Basic, UNTIL in Pascal), wobei die Schleife immer mindestens einmal durchlaufen wird.

Für den Abruf von Unterprogrammen gibt es schließlich ein eigenes Symbol *(Abb. 3.3.2.8),* ein ovales Feld mit dem Namen des Unterprogramms.

Wenn wir das im Kapitel über Flußdiagramme verwendete Beispiel, eine Disk-Datei zu drucken, mit einem Struktogramm realisieren, so liegt zunächst die Mög-

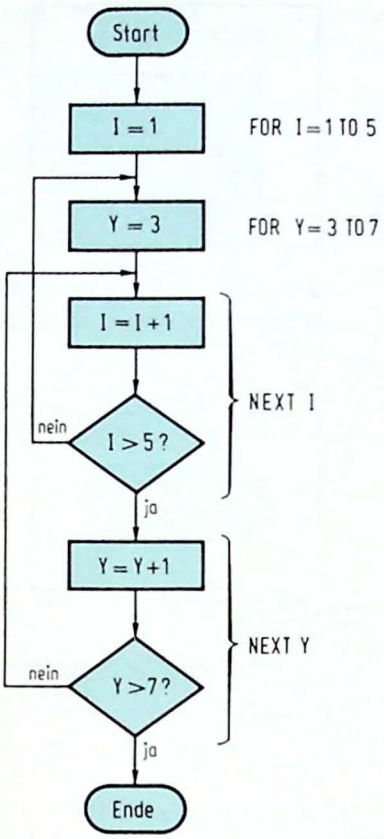


Abb. 3.3.2.1 Falsche Verschachtelung von Schleifen in einem Programm

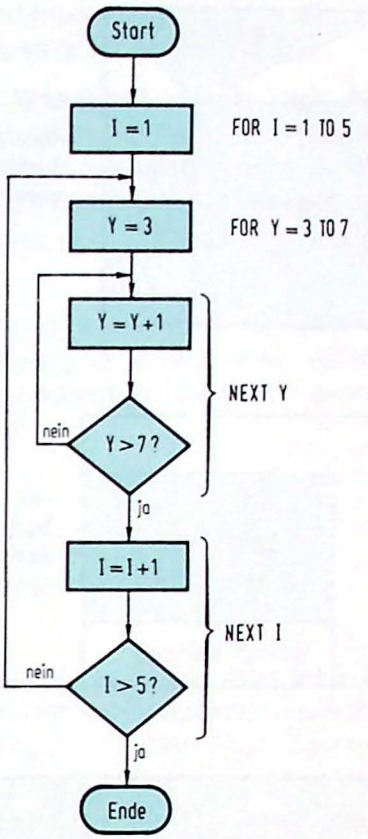


Abb. 3.3.2.2 So muß die Verschachtelung richtig aussehen

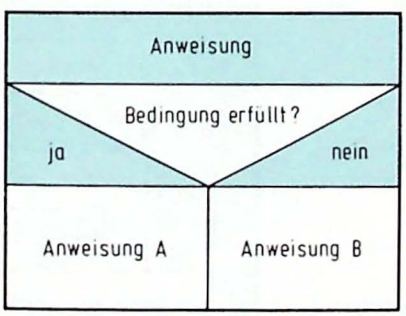


Abb. 3.3.2.3 Bedingungs-Abfrage in einem Struktogramm (Nassi-Shneiderman-Diagramm)

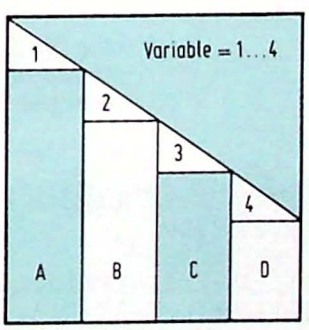


Abb. 3.3.2.4 Dieses Struktogramm-Element entspricht der CASE-Anweisung in Pascal bzw. dem Befehl ON I GOSUB in Basic

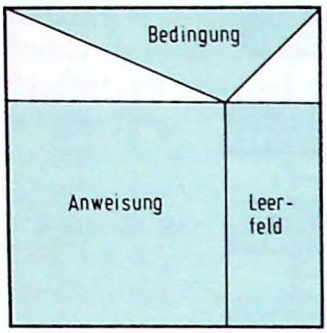


Abb. 3.3.2.5
Ausführen einer Anweisung, wenn die Bedingung erfüllt ist

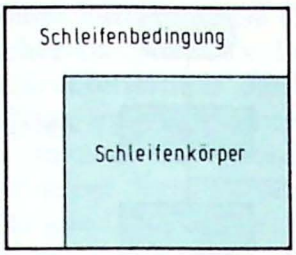


Abb. 3.3.2.6 Programmschleife mit einer am Anfang stehenden Abbruch-Bedingung

Abb. 3.3.2.7
Schleife mit einer am Ende stehenden Abbruch-Bedingung

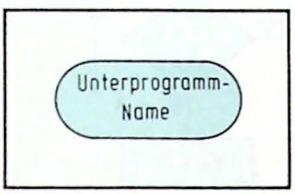


Abb. 3.3.2.8
Aufruf eines Unterprogramms

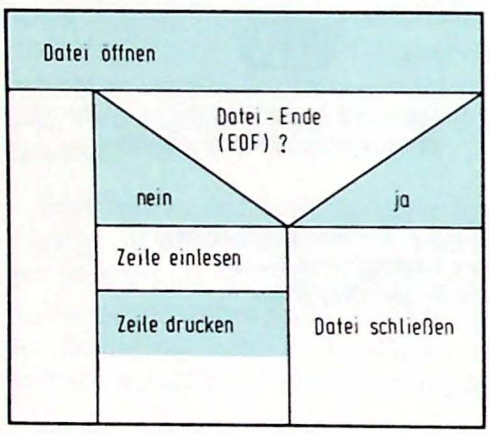


Abb. 3.3.2.9
Struktogramm-Fragment zum Ausdrucken einer Disk-Datei

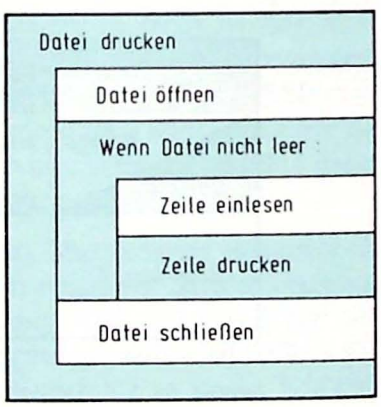


Abb. 3.3.2.10 Das fertige Struktogramm, jedoch mit einer bedingten Schleife realisiert

```
10 REM DATEI "ADR" AUSDRUCKEN
20 OPEN"I",1,"ADR"
30 WHILE NOT EOF(1)
40 INPUT#1,Z$:LPRINT Z$:WEND
50 CLOSE 1:END
```

Abb. 3.3.2.11
MBasic-Programm mit WHILE-Anweisung

lichkeit nach *Abb. 3.3.2.9* nahe, bei der die Lese/Drucke-Schleife erst nach einer Abfrage begonnen wird, ob das Dateiende erreicht ist (EOF = End of File).

Noch einfacher geht es allerdings mit dem Schleifen-Konstrukt nach *Abb. 3.3.2.10:* Am Anfang der Schleife Lesen/Drucken steht die Abbruch-Bedingung „Wenn Datei nicht leer". Wenn die Datei dagegen leer ist, wird sie geschlossen, das Programm ist beendet. Erwähnt werden muß noch, daß der Name des gesamten Programms in einem zusätzlichen Kasten erscheint, der es komplett umschließt (ähnlich der PROGRAM-Anweisung in Pascal!).

Aus diesem letzten Struktogramm ergibt sich fast automatisch eine neue, im Vergleich zum letzten Kapitel einfachere Realisierung in MBasic unter Verwendung der Anweisungen WHILE und WEND, die allerdings leider nicht in allen Basic-Dialekten zur Verfügung stehen *(Abb. 3.3.2.11).*

3.3.3 Syntax-Diagramme

Die bisher besprochenen grafischen Darstellungen verdeutlichen, was eine Folge von Befehlen (ein Programm) tut, was sie bewirkt. Syntax-Diagramme geben darüber keine Auskunft, sondern definieren lediglich die Schreibweise bestimmter Befehle.

Bei Syntax-Diagrammen gibt es nur drei Symbole. Kleine Kreise markieren einzelne Zeichen, z.B. ein Komma zur Trennung zweier Parameter nach einem Befehl. Ovale Symbole enthalten den Namen einer neu zu definierenden Zeichenfolge. Eckige Kästchen dagegen nehmen auf bereits vorher definierte Ausdrücke oder Befehle bezug. Die Symbole werden untereinander durch Flußlinien verbunden.

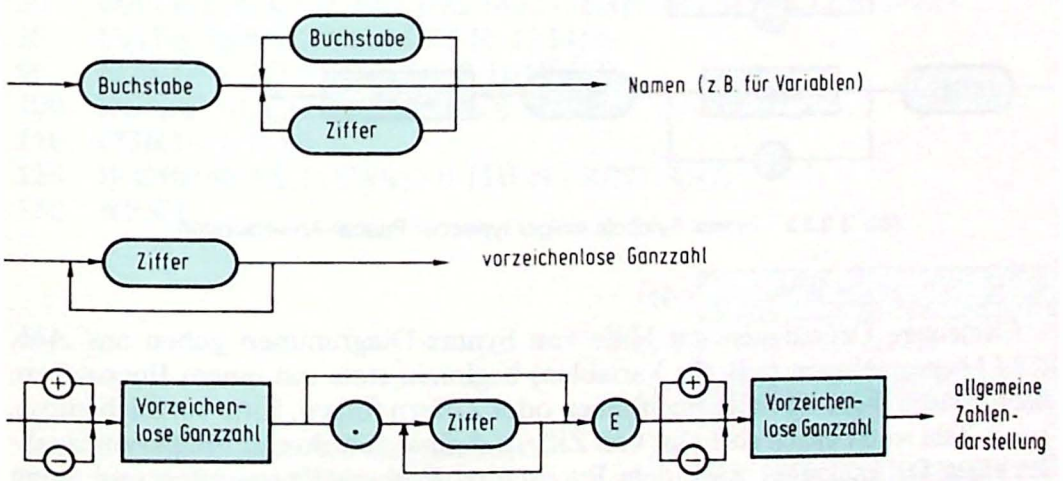


Abb. 3.3.3.1 Syntax-Symbole für Zahlen

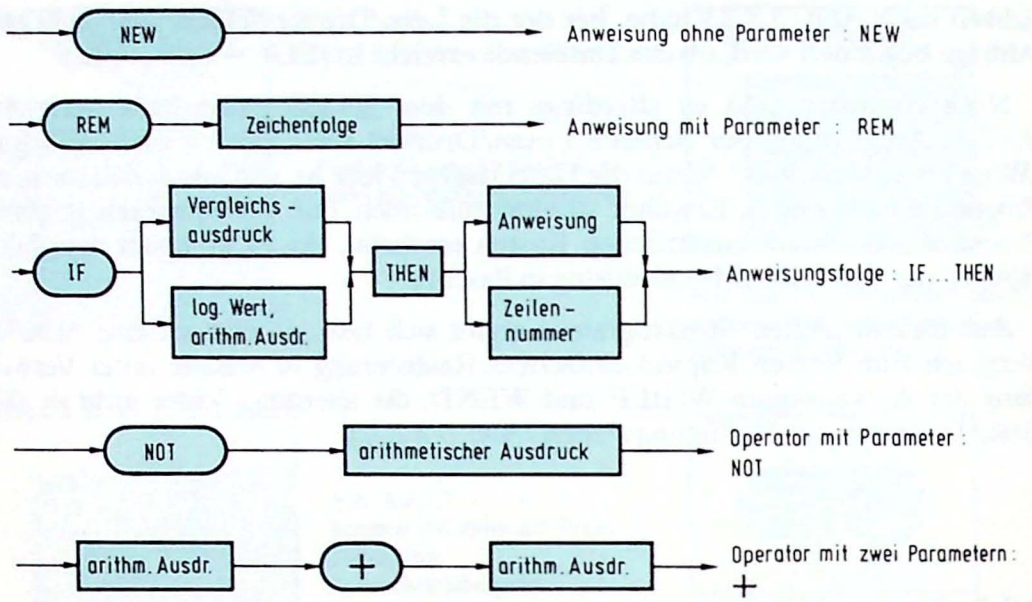


Abb. 3.3.3.2 Beispiele für Syntax-Symbole einiger Basic-Befehle

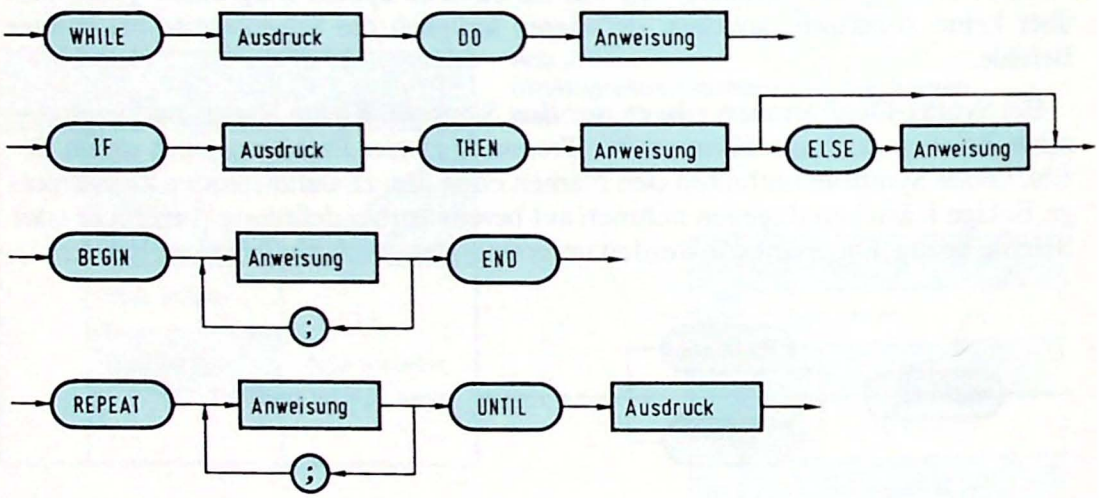


Abb. 3.3.3.3 Syntax-Symbole einiger typischer Pascal-Anweisungen

Elementare Definitionen mit Hilfe von Syntax-Diagrammen gehen aus *Abb. 3.3.3.1* hervor: Namen (z.B. für Variablen) beginnen stets mit einem Buchstaben, dann können (beliebig viele) Buchstaben oder Ziffern folgen. Eine vorzeichenlose, ganze Zahl wird einfach als Folge von Ziffern definiert, während Fließkommazahlen einen Dezimalpunkt, nach dem Buchstaben E einen Exponenten und je ein Vorzeichen für Mantisse (nicht gezeichnet) und Exponent enthalten können.

Die Syntax-Diagramme für einige Basic-Befehlsstrukturen sind in *Abb. 3.3.3.2* gezeigt. Der Einfachkeit halber sind hier nicht alle Zwischenschritte z.B. zur Definition arithmetischer Ausdrücke gezeigt.

Typische Pascal-Befehle sind in *Abb. 3.3.3.3* als Struktogramme aufgeführt. Besonders bei Anweisungen wie BEGIN und END wird deutlich, daß solche Grafiken wirklich nur die Schreibweise, nicht aber die Wirkung der Befehle darstellen: Denn die Anweisung zwischen BEGIN und END wird keineswegs wiederholt. Vielmehr können, durch Strichpunkte getrennt, mehrere Anweisungen zwischen BEGIN und END als einzige Verbund-Anweisung zusammengefaßt werden.

3.3.4 Datenstrukturen

Auch die Verkettung von Daten z.B. in einer Datenbank kann man sich am besten an Hand von Grafiken vorstellen. Deshalb soll dieses Thema ebenfalls in diesem Kapitel besprochen werden. Als Beispiel dient dabei zunächst eine einfache Kartei mit Telefonnummern, die der Übersichtlichkeit halber nur fünf Namen umfaßt.

Planare Strukturen

Abb. 3.3.4.1 zeigt, wie man so etwas programmtechnisch lösen könnte. Jeder Name und die dazugehörige Telefonnummer erhält sozusagen eine Karteikarte, beispielsweise ein Element einer Array-Variablen. Wenn man nun später die Telefonnummer von Gabi herausfinden will, ohne ihren Nachnamen zu wissen, so könnte man für den Suchvorgang in Basic den Befehl INSTR verwenden, der ermittelt, ob ein String in einem anderen enthalten ist. Ein Versuchsprogramm hierzu könnte so aussehen:

```
10   DATA HANS MEIER; 123456, MARINA MEIER; 123456
20   DATA GABI MUELLER; 54321, ERHART MUELLER; 54321
30   DATA ROBERT HACKER; 123456
90   FOR I=1 TO 5:READ A$(I):NEXT
100  INPUT "SUCHWORT:";SW$
110  FOR I=1 TO 5
120  IF INSTR(A$(I),SW$)>0 THEN PRINT A$(I)
130  NEXT
```

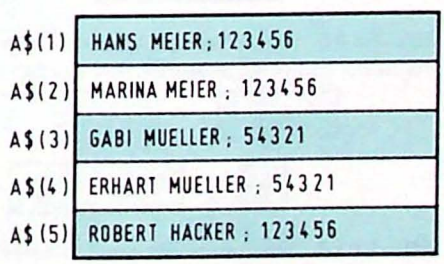


Abb. 3.3.4.1 Struktur einer planaren Datenbank. Ein einziges Array namens A$ beinhaltet sämtliche Daten zeilenweise

1	HANS
2	MARINA
3	GABI
4	ERHART
5	ROBERT

1	MEIER
2	MEIER
3	MUELLER
4	MUELLER
5	HACKER

1	123456
2	123456
3	54321
4	54321
5	123456

Abb. 3.3.4.2
Unterteilen der Daten in Vornamen, Namen und Telefonnummern

Dieses Basic-Programm liest die Namen und Telefonnummern aus den DATA-Zeilen in die Array-Variable A$ ein. Ist das gesuchte Wort (z.B. Gabi) in einer der Elemente von A$ enthalten, so wird dieses Element (also Name und Telefonnummer) ausgegeben. Dies ist eine einfache Realisierung einer sogenannten „planaren" Datenstruktur, d.h. die zu suchenden und die auszugebenden Daten liegen in derselben „Ebene".

Eine kleine Abwandlung dieser planaren Struktur ist in *Abb. 3.3.4.2* gezeigt. Vorname, Name und Telefonnummer sind jeweils eigenen Array-Variablen zugeordnet (z.B. V$(I), N$(I) und TN%(I)) oder auch einem einzigen zweidimensionalen Array (z.B. A$(I,J), wobei J=1 den Vornamen anwählt, J=2 den Nachnamen und J=3 die Telefonnummer als String). Diese Struktur hat den Vorteil, daß man für die Suche nach einem bestimmten Vornamen nicht mehr die gesamten Daten durchsuchen muß, sondern nur noch diejenigen Strings, die wirklich nur die Vornamen enthalten.

Verkettete Listen

Vielleicht ist Ihnen schon aufgefallen, daß mehrere Nachnamen und Telefonnummern in unserem Beispiel gleich sind. Hans und Marina wohnen zusammen, ebenso Gabi und Erhart. Herr Hacker wohnt bei den Meiers zur Untermiete. Mit einem Trick läßt sich daraus Profit schlagen, nämlich Speicherplatz sparen, wie *Abb. 3.3.4.3* zeigt: Jedem Vornamen-Feld wird ein Zeiger zugeordnet, der auf die Num-

	A$(I,1)	N%(I,1)
1	HANS	1
2	MARINA	1
3	GABI	2
4	ERHART	2
5	ROBERT	3

	A$(I,2)	N%(I,2)
1	MEIER	1
2	MUELLER	2
3	HACKER	1
4		
5		

	A$(I,3)	
1	123456	
2	54321	
3		
4		
5		

Abb. 3.3.4.3 Durch Verwendung von Zeigern kann man Speicherplatz sparen, da gleichartige Elemente nur noch einmal gespeichert werden müssen

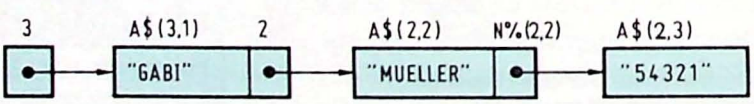


Abb. 3.3.4.4 So kann ein Programm die Telefonnummer von Gabi Müller herausfinden

mer des zugehörigen Nachnamen-Feldes weist. Ebenso gehört zu jedem Nachnamen ein Zeiger auf die zugehörige Telefonnummer. Dadurch ist es nicht mehr nötig, ohnehin gleiche Namen und Telefonnummern mehrmals im Speicher abzulegen. Lediglich für die Zeiger (z. B. eine Integer-Array-Variable N%(I,J) mit J = 1...2) muß zusätzlicher Speicherplatz reserviert werden.

Das Prinzip ist nochmals in *Abb. 3.3.4.4* veranschaulicht. Der Vorname Gabi wurde im Vornamen-Feld mit dem Index 3, also in A$(3,1), durch einfachen Stringvergleich gefunden. Nun sieht der Rechner im Zeigerfeld N%(3,1) nach und findet dort die Zahl 2. Das bedeutet, der zugehörige Nachname findet sich im Nachnamen-Feld mit der Nummer 2, also in A$(2,2). Im zugehörigen Zeiger N%(2,2) steht der Verweis auf die Telefonnummer von Gabi Müller, nämlich auf das Nummernfeld Nr. 2 entsprechend A$(2,3). Eine solche Datenstruktur nennt man „verkettete Liste".

Binärbäume

Eine weitere Möglichkeit des Auffindens bestimmter Daten wurde im Kapitel über die Programmiersprache Basic bereits an Hand eines Beispielsprogramms gezeigt (Tiere raten). Die gewünschte Information erhält man durch wiederholte Ja/Nein-Abfragen und dementsprechende Verzweigung in einer Struktur, die den Wurzeln eines Baumes oder auch, wenn man den Baum umdreht, seinen Ästen entspricht *(Abb. 3.3.4.5)*.

Das Prinzip ähnelt dem der verketteten Liste. Allerdings gehört hier zu jedem Element nicht ein Zeiger, sondern es sind jeweils zwei — einer für das Element, das bei „Nein" angewählt werden soll, und einer für das bei „Ja".

Die erste Frage könnte z. B. wie in dem gerade erwähnten Basic-Programm lauten: „Ist es ein Säugetier?" Bei der Antwort Ja könnte man zur nächsten Frage „Lebt es im Urwald?" gehen, bei Nein „Kann es schwimmen?" fragen, und so wei-

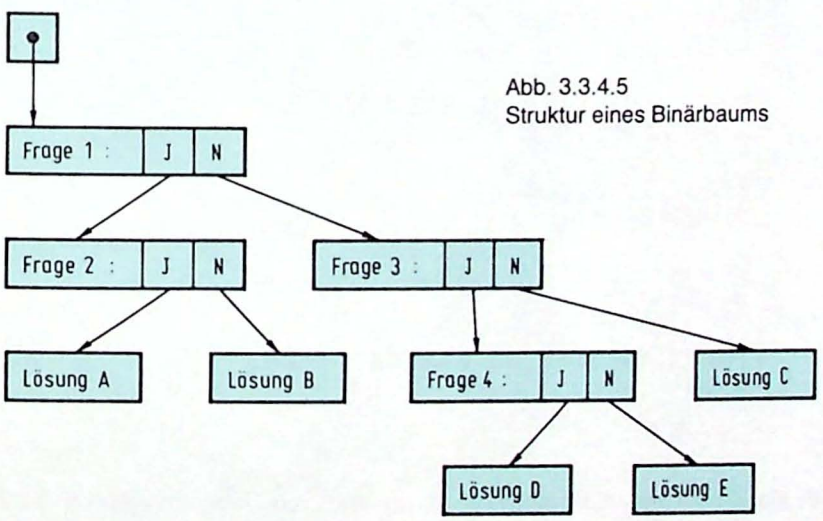


Abb. 3.3.4.5
Struktur eines Binärbaums

```
>RUN

LADEN,SPEICHERN,ANZEIGEN,DIALOG,EDIT.?L

FILENAME:TIERE

LADEN,SPEICHERN,ANZEIGEN,DIALOG,EDIT.?A

ZAHL DER EINTRAEGE:55

FRAGE 0: LEBT ES IM WASSER?
N:1                 J:2

FRAGE 1: IST ES EIN SAEUGETIER?
N:4                 J:3

FRAGE 2: IST ES EIN SAEUGETIER?
N:5                 J:6

FRAGE 3: IST DER SCHWANZ AUSGEPRAEGT?
N:41                J:12

FRAGE 4: KANN ES FLIEGEN?
N:34                J:9

FRAGE 5: IST ES EIN FISCH?
N:7                 J:8

FRAGE 6: IST ES SEHR GROSS?
N:DELPHIN           J:WAL

FRAGE 7: HAT ES VIER BEINE?
N:22                J:27

FRAGE 8: FRISST ES FISCHE?
N:45                J:46

FRAGE 9: HAT ES ZWEI BEINE?
N:10                J:19

FRAGE 10: KANN ES STECHEN?
N:STUBENFLIEGE      J:11

FRAGE 11: PRODUZIERT ES ESSBARES?
N:WESPE             J:BIENE

FRAGE 12: HAT ES EINEN RUESSEL?
N:13                J:49

FRAGE 13: IST ES EIN RAUBTIER?
N:16                J:14
```

Abb. 3.3.4.6 Beispiel für einen Binärbaum für das Ermitteln von Tierarten (vgl. Abb. 3.2.3.5)

```
FRAGE 14: HAT ES EIN GEMUSTERTES FELL?
N:LOEWE                 J:15

FRAGE 15: IST ES EIN PUNKTEMUSTER?
N:TIGER                 J:LEOPARD

FRAGE 16: IST ES EIN HAUS- O.NUTZTIER?
N:25                    J:17

FRAGE 17: IST ES EINE HUNDEART?
N:20                    J:18

FRAGE 18: IST ES ZUR BEWACHUNG GEEIGNET?
N:24                    J:SCHAEFERHUND

FRAGE 19: TRINKT ES UNMAESSIG?
N:28                    J:55

FRAGE 20: HAT ES HOERNER?
N:HAUSKATZE             J:21

FRAGE 21: IST ES MAENNLICH?
N:KUH                   J:STIER

FRAGE 22: HAT ES ZWEI BEINE?
N:KREBS                 J:23

FRAGE 23: HAT ES EINEN LANGEN HALS?
N:WILDENTE              J:SCHWAN

FRAGE 24: HAT ES KURZE BEINE?
N:PUDEL                 J:DACKEL

FRAGE 25: LEBT ES IM URWALD?
N:FELDMAUS              J:26

FRAGE 26: LAEUFT ES OFT AUFRECHT?
N:PAVIAN                J:SCHIMPANSE

FRAGE 27: IST ES MEIST GRUEN?
N:WASSERSCHILDKROETE J:KROKODIL

FRAGE 28: IST ES GROSS?
N:29                    J:30

FRAGE 29: HACKT ES LOECHER IN BAEUME?
N:33                    J:BUNTSPECHT

FRAGE 30: FRISST ES LEICHEN?
N:31                    J:AASGEIER

FRAGE 31: LEBT ES IM GEBIRGE?
N:32                    J:STEINADLER

FRAGE 32: KANN ES SCHNELL LAUFEN?
N:STORCH                J:STRAUSSENVOGEL
```

zu Abb. 3.3.4.6

```
FRAGE 33: IST ES EIN SINGVOGEL?
N:47                J:MEISE

FRAGE 34: HAT ES ZAEHNE?
N:38                J:35

FRAGE 35: HAT ES VIER BEINE?
N:36                J:37

FRAGE 36: KLAPPERT ES?
N:KREUZOTTER        J:KLAPPERSCHLANGE

FRAGE 37: IST ES GROSS?
N:SALAMANDER        J:SAURIER

FRAGE 38: HAT ES BEINE?
N:WURM              J:39

FRAGE 39: IST ES GIFTIG?
N:40                J:VOGELSPINNE

FRAGE 40: HAT ES MEHR ALS SECHS BEINE?
N:48                J:TAUSENDFUESSLER

FRAGE 41: LAUEFT ES AUF VIER FUESSEN?
N:MENSCH            J:42

FRAGE 42: HAT ES EIN WEISSES FELL?
N:43                J:EISBAER

FRAGE 43: IST ES SEHR GEFAEHRLICH?
N:44                J:GRIZZLY

FRAGE 44: HAT ES EINEN LANGEN HALS?
N:52                J:KAENGURUH

FRAGE 45: IST ES EIN SUESSWASSERFISCH?
N:FLUNDER           J:KARPFEN

FRAGE 46: IST ES GROSS?
N:HECHT             J:HAI

FRAGE 47: IST ES NUETZLICH?
N:TAUBE             J:HUHN

FRAGE 48: HAT ES EINEN SCHNABEL?
N:HAUSSPINNE        J:DONALD DUCK

FRAGE 49: IST ES ROSA?
N:51                J:50

FRAGE 50: IST ES HUEBSCH?
N:HAUSSCHWEIN       J:MISS PIGGY
```

zu Abb. 3.3.4.6

```
FRAGE 51: LEBT ES IN EUROPA?
N:ELEFANT              J:WILDSCHWEIN

FRAGE 52: LEBT ES IN DEUTSCHLAND?
N:54                   J:53

FRAGE 53: HAT ES EIN GEWEIH?
N:REH                  J:HIRSCH

FRAGE 54: HAT ES EIN GEWEIH?
N:SCHWARZBAER          J:ELCH

FRAGE 55: HACKT ES AUF BAEUME EIN?
N:SCHNAPSDROSSEL       J:SCHLUCKSPECHT

LADEN,SPEICHERN,ANZEIGEN,DIALOG,EDIT.?Q   zu Abb. 3.3.4.6
```

ter. Am Ende des Baumes gelangt man schließlich zu den gesuchten Tiernamen, wie Goldfisch, Schimpanse usw.

In dem erwähnten Basic-Programm werden alle Fragen im Array F$ und entweder Verweiszeiger oder Tiernamen (Antworten) im Array N$ gespeichert: Beginnt N$ mit einer Ziffer, so handelt es sich um einen Verweiszeiger auf die nächste Frage in F$, andernfalls ist man am Baumende angelangt, und das Programm fragt, ob die gefundene Antwort richtig ist. Sollte das nicht der Fall sein, wird der Baum mit einem neuen Tiernamen ergänzt, indem die vom Benutzer einzugebende Unterscheidungs-Frage in F$ und das neue Tier in N$ eingefügt wird. Wie so ein Baum intern aussieht, läßt sich mit dem Kommando A anzeigen; ein Beispiel-Ausdruck ist in *Abb. 3.3.4.6* zu finden. Dabei ist jeweils eine Frage zusammen mit den beiden möglichen Antworten J und N aufgeführt; eine Zahl in einem Antwortfeld stellt einen Verweis auf eine andere Frage dar.

Das Prinzip des binären Baums ist keineswegs auf ein Frage- und Antwortspiel zwischen Rechner und Benutzer begrenzt. Der Rechner kann einen Baum auch an Hand von ihm selbst zu beantwortender Kriterien durchlaufen. So ist es z.B. möglich, auf bestimmte Namen einer Adressenkartei sehr schnell und effizient zuzugreifen, indem man sie nach dem Alphabet in einem Binärbaum ordnet. Die erste Frage entspricht dann z.B., ob der Name mit A bis M beginnt. Lautet die Antwort Ja, prüft der Rechner weiter, ob der Anfangsbuchstabe zwischen A und F liegt, und so weiter. Auf diese Weise wird der Bereich immer weiter eingeengt, und am Ende des Baums ist genau ein Name.

4 Mikrocomputer-Betriebssysteme

Als Betriebssysteme im weiteren Sinn bezeichnet man jene elementare Software, die im Computer auf jeden Fall immer vorhanden sein muß, um die elementaren Aufgaben wie Tastatur-Eingabe, Bildschirm-Ausgabe und Datentransfer von und auf Massenspeicher (z.B. Floppy) zu erledigen. Im engeren Sinne meint man jedoch bei Mikrocomputern jene Programmpakete, die alle notwendigen Routinen zur Verwaltung von Dateien auf Floppies oder Festplatten enthalten.

Es gibt sowohl Betriebssysteme, die relativ hardware-unabhängig sind und auf Mikrocomputern der unterschiedlichsten Hersteller laufen (CP/M, MS-DOS, Unix, OS-9 u.a.) als auch solche, die von vornherein auf die Hardware eines bestimmten Herstellers angewiesen sind (Commodore, Apple u.a.). Manchmal sitzt das Betriebssystem im ROM, in den meisten Fällen befindet sich im ROM jedoch nur ein kurzes Urlade-Programm, das das eigentliche Betriebssystem beim Einschalten des Rechners automatisch in einen reservierten Bereich des Arbeitsspeichers lädt.

4.1 Standard-Betriebssysteme

Manche Betriebssysteme wurden dadurch, daß sie von mehreren Hardware-Herstellern verwendet wurden, zu De-facto-Standards (d.h. sie wurden keineswegs von irgendeiner Behörde zur Norm erhoben). Dies bedeutet, daß die Anwender-Programme, die für diese Betriebssysteme geschrieben worden sind, ebenfalls auf Rechnern unterschiedlicher Hersteller laufen — vorausgesetzt, diese Programme verwenden ausschließlich die vom Betriebssystem her standardisierten Einsprungstellen und nicht etwa spezifische Merkmale eines Rechners.

Die Rechner-Hardware kann durchaus recht unterschiedlich ausgelegt sein, beispielsweise können auch bei betriebssystem-kompatiblen Computern die Peripherie-Bausteine an unterschiedlichen Adressen sitzen. Das Betriebssystem muß durch Anpassung bestimmter Sprungvektoren an den einzelnen Rechner erst angepaßt werden.

Ferner ist die Verwendung des gleichen Betriebssystems bei mehreren Rechnertypen nicht unbedingt damit gleichzusetzen, daß die Disketten, die auf einem be-

schrieben wurden, mit dem anderen ohne weiteres lesbar sind. Wichtige Parameter, wie z.B. Anzahl der Spuren und Sektoren auf der Diskette sowie die Wahl des Interleaving-Faktors (siehe Kapitel über Floppy-Disk-Aufzeichnung!), werden durch das Betriebssystem nicht festgelegt und können nach wie vor vom Hardware-Hersteller frei gewählt werden. Hinzu kommt, daß die meisten üblichen Standard-Betriebssysteme lediglich Textdateien, nicht aber Grafik unterstützen, so daß bei Grafik-Anwendungen ein gleiches Betriebssystem keinerlei Aussage über die Kompatibilität zweier Computer darstellt.

4.1.1 CP/M

Das erste weiter verbreitete Standard-Betriebssystem für Mikrocomputer mit der CPU 8080 wurde von seinem Entwickler Gary Kildall bei Digital Research CP/M (Control Program for Microcomputers) genannt. Da der später von Zilog herausgebrachte Prozessor Z80 den 8080-Befehlssatz als Untermenge ausführen kann, ist CP/M-80 (so nennt man es heute zur Unterscheidung von den Versionen für andere CPU-Familien) für 8080-, 8085- und Z80-Rechner geeignet. Die weitaus meisten CP/M-Rechner arbeiten heute mit der CPU Z80. Für dieses Betriebssystem ist daher sehr viel Anwender-Software erhältlich, vor allem auch preiswerte.

Die am weitesten verbreitete Version ist bisher CP/M 2.2, während mit CP/M 3.X ein Paket zur Verfügung steht, das das „Banking" bei mehr als 64 KByte RAM unterstützt, selbst allerdings auch mehr Platz im Speicher belegt. Im folgenden wird zunächst CP/M 2.2 besprochen.

CP/M wird auf einer für einen bestimmten Rechnertyp angepaßten Diskette von Distributoren geliefert. Diese Diskette enthält eine ganze Reihe von Programmen, wie das eigentliche Betriebssystem selbst, einen 8080-Assembler (die zusätzlichen Z80-Codes können damit nicht generiert werden!), einen einfachen Texteditor z.B. zum Erstellen von Assembler-Sourcetexten sowie einige Hilfsprogramme wie etwa zum Formatieren und Kopieren von Disketten.

Innerhalb des eigentlichen Betriebssystems, das immer im Arbeitsspeicher (oder, seltener, im ROM) steht, unterscheidet man drei Programmteile:

1. BIOS	Basic I/O System	Ein-/Ausgabesystem (hardwareabhängig)
2. BDOS	Basic Disk Operating System	Routinen zur Bedienung von Disk-Laufwerken
3. CCP	Console Command Processor	Terminal-Kommando-Interpreter

Das Wort „Basic" hat hier nichts mit der gleichlautenden Programmiersprache zu tun, sondern heißt soviel wie „grundlegend". Den verbleibenden Platz im Ar-

beitsspeicher nennt man TPA (Transient Program Area); er steht für Assembler, Interpreter, Compiler und Anwenderprogramme zur Verfügung. Der Arbeitsspeicher muß für ein CP/M-System mindestens 20 KByte umfassen, bei den neueren CP/M-Computern sind jedoch gewöhnlich 64 KByte oder (bei Bank-Umschaltung) auch mehr vorhanden.

Banking und Booten

Da der RAM-Inhalt eines Computers beim Einschalten grundsätzlich undefiniert ist, ist in jedem Fall ein ROM erforderlich, um für eine korrekte Initialisierung zu sorgen und dann das Betriebssystem zu laden. CP/M verwendet nun allerdings den Bereich hex 0000...00FF, in dem bei 8080- und Z80-Systemen das ROM mit der Reset-Routine liegen muß, für verschiedene Parameter, so daß dort RAM vorhanden sein muß. Dieser Widerspruch muß mit Hilfe einer Banking-Logik aufgelöst werden. *Abb. 4.1.1.1* zeigt die Adressen-Verteilung in einem Rechner mit 64 KByte RAM vor und nach dem Laden des Betriebssystems, was man übrigens „Boot-Vorgang" nennt.

Im einzelnen passiert folgendes. Zunächst ist im Adressenbereich ab 0000 das ROM eingeblendet, so daß bei einem Reset das in ihm liegende Initialisierungs-Programm abgearbeitet wird. Dieses Programm kann relativ kurz sein, weil es nichts weiter tut, als den restlichen EPROM-Inhalt z.B. ab der RAM-Adresse F000 zu kopieren und dann an eben jene Adresse zu springen. Einer der ersten Befehle dort schaltet über die Banking-Logik das ROM bzw. EPROM nun aus und das RAM ab Adresse 0000 ein, so daß 64 KByte RAM zur Verfügung stehen, von denen die obersten 4 KByte (F000...FFFF) vom umkopierten EPROM-Inhalt belegt sind. Das dort weiter ablaufende Programm muß nun lediglich die Spur 0 von

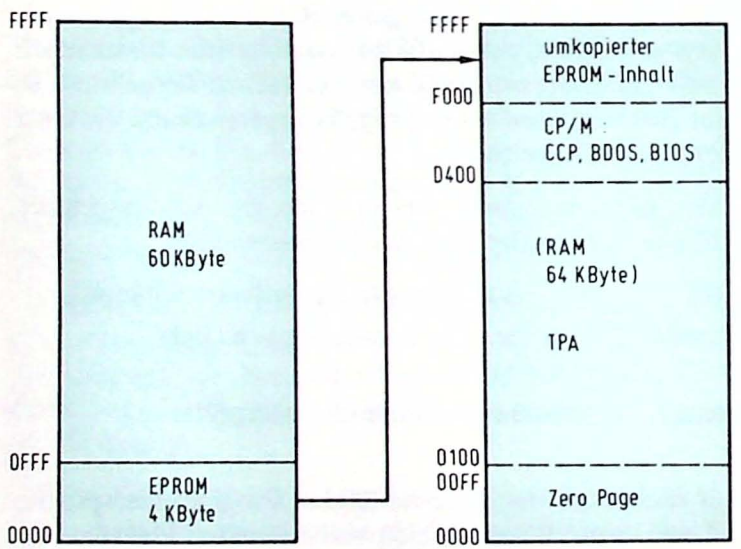


Abb. 4.1.1.1
Speicherverteilung eines CP/M-Computers beim Einschalten (links) und nach dem Laden des Betriebssystems (rechts)

der Diskette laden, auf der das CP/M-Betriebssystem steht. Dazu ist ein sogenanntes Urlader- oder Bootstrap-Programm im EPROM erforderlich.

Die Adresse F000 diente hier nur als Beispiel. Bei weniger RAM muß sie tiefer liegen, bei trickreicher Bank-Umschaltung kann man nach Abschluß des Boot-Vorgangs sogar das gesamte RAM von 64 KByte völlig frei halten. Je nach freiem RAM ergibt sich folgende Adressenbelegung:

KByte	BIOS	BDOS	CCP
20	4A00	3C00	3400
24	5A00	4C00	4400
32	7A00	6C00	6400
40	9A00	8C00	8400
48	BA00	AC00	A400
56	DA00	CC00	C400
60	EA00	DC00	D400
62	F200	E400	DC00
64	FA00	EC00	E400

Der freie Bereich (die TPA) beginnt stets bei der Adresse hex 0100. In der „Zero Page" sind von CP/M folgende Adressen belegt:

von	bis	Inhalt
0000	0002	Sprung zur Warmstart-Routine
0003	0003	I/O-Byte (Intel-Standard)
0004	0004	Aktuelles Laufwerk (0...15)
0005	0007	Sprung zum BDOS
0008	0037	Interrupts 1...6 (nicht verwendet)
0038	003A	Von DDT/SID für Breakpoints verwendet
003B	003F	nicht verwendet
0040	004F	CBIOS-Arbeitsbereich, normalerweise nicht verwendet
0050	005B	nicht verwendet
005C	007C	Bereich für File-Control-Block (FCB)
007D	007F	Verlängerung des FCB bei Random-Zugriff
0080	00FF	DMA-Bereich und Kommandozeilen-Puffer

Das I/O-Byte

Das I/O-Byte stellt ein Bitmuster dar, das einem bestimmten logischen Gerät ein physikalisches zuordnen kann, beispielsweise, um alle Ausgaben, die als „Konsole" spezifiziert sind, auf den Bildschirm (CRT) zu leiten. Dazu dienen jeweils zwei Bits des I/O-Byte:

Bit-Nr.	Log. Ausgabekanal	Physikalisches Gerät
7, 6	LIST (Auflisten)	00 TTY (Teletype) 01 CRT (Cathode Ray Tube) 10 LPT (Line Printer) 11 UL1 (User List Device 1)
5, 4	PUNCH (Lochstreifen)	00 TTY (Teletype) 01 PTP (Punch Tape Puncher) 10 UP1 (User Puncher 1) 11 UP2 (User Puncher 2)
3, 2	READER (Lochstreifen)	00 TTY (Teletype) 01 PTR (Punch Tape Reader) 10 UR1 (User-defined Reader 1) 11 UR2 (User-defined Reader 2)
1, 0	CONSOLE (Konsole)	00 TTY (Teletype) 01 CRT (Cathode Ray Tube) 10 BAT (Batch: READER = Eing., LIST = Ausg.) 11 UC1 (User defined Console 1)

Die einzelnen Abkürzungen haben im Deutschen folgende Bedeutung:

TTY	ASCII-Fernschreiber
PTP	Lochstreifen-Stanzer
PTR	Lochstreifen-Lesegerät
LPT	Zeilenorientierter Drucker
CRT	Bildschirm
Uxx	Vom Benutzer definierbare Schnittstelle (durch BIOS-Zeiger)

Der File-Control-Block

Der FCB stellt alle für einen Disk-Zugriff benötigten Parameter zur Verfügung. Für sequentielle Zugriffe (d.h. ein Datenfeld nach dem anderen wird aufeinanderfolgend beschrieben oder gelesen) umfaßt er 33 Byte, für Random-Zugriffe (ein beliebiges Element einer Datei wird gelesen, ohne die vorher geschriebenen auch lesen zu müssen) 36 Byte. Er ist wie folgt aufgebaut:

Byte	Inhalt
0	0 = voreingestellte Laufwerk-Nr., 1 ... 16 = A ... P
1 ... 8	Dateiname in Großbuchst., Bit 7 = 0
9 ... 11	Dateityp in Großbuchst., Bit 7 = 1 bedeutet bei Byte 9: Datei schreibgeschützt; bei Byte 10: SYS-Datei, wird bei DIR nicht aufgeführt
12 ... 15	vom System verwendet
16 ... 31	vom System für zweiten FCB reserviert
32	Nummer des gerade gelesenen Blocks
33 ... 35	Blocknummer für Random-Zugriffe

Der FCB wird für jede Datei auf der Diskette im Inhaltsverzeichnis geführt. Wird eine Datei durch einen Schreibvorgang verändert, wird selbstverständlich auch der zugehörige FCB korrigiert, sobald die Datei geschlossen wird (CLOSE in Basic).

Der Dateityp wird auch als Extension (Erweiterung des Dateinamens) bezeichnet. Er steht durch einen Punkt getrennt als Kombination aus bis zu drei Buchstaben hinter dem eigentlichen Dateinamen. Beispielsweise sind BENCH.BAS, SOURCE.ASM oder DATEN gültige Dateinamen (die Extension darf also auch fehlen). Folgende Dateitypen sind unter CP/M üblich:

ASM	Assembler-Sourcetext
BAK	Sicherheitskopie einer Datei
BAS	Basic-Programm
COM	CP/M-Kommando-Datei (z.B. STAT.COM)
HEX	Maschinenprogramm im Intel-Hex-Format
INT	Zwischencode
$$$	Vorübergehend auf Disk zwischengespeicherte Daten
PAS	Pascal-Sourcetext
PRN	Drucker-Listing-Datei
REL	Verschiebbares Maschinensprache-Modul
SYM	Datei mit Assembler-Symboltabelle

Die Bedienung von CP/M

Nach dem Boot-Vorgang meldet sich das Betriebssystem mit dem Laufwerk A (das ist das zunächst voreingestellte Disk-Laufwerk) und einem Größer-Zeichen als „Prompt-Symbol", also so:

```
A>
```

Der Kommando-Interpreter (CCP) übernimmt in CP/M die Kommunikation mit dem Benutzer. Er liest alle Kommandos zeilenweise, d.h. bis zum Abschluß einer Zeile mit Return, in einen Puffer, wandelt alle Klein- in Großbuchstaben um und decodiert die Kommandos. (Die automatische Umwandlung in Großbuchstaben kann zu Problemen führen: Erzeugt man beispielsweise unter MBasic eine Datei, deren Namen aus Kleinbuchstaben besteht, dann kann man sie von CP/M aus weder listen noch löschen oder verändern!)

Unter CP/M unterscheidet man sogenannte interne und externe Kommandos. Die zur Ausführung der „internen" Kommandos (z.B. DIR) nötigen Routinen sind im Betriebssystem-Kern selbst vorhanden, also dauernd im Arbeitsspeicher verfügbar. Bei externen Kommandos (z.B. STAT) sucht der Kommando-Interpreter eine Datei mit dem jeweiligen Namen und der Extension „.COM" und lädt sie in den TPA-Bereich (d.h. ab Adresse hex 0100).

Zunächst jedoch eine ganz wichtige Regel für CP/M 2.2: Im Gegensatz zu praktisch allen anderen Betriebssystemen ist es unbedingt erforderlich, CTRL-C zu drücken, wenn man eine andere Diskette in ein Laufwerk steckt. Nur dann wird nämlich der noch von der alten Disk stammende File-Control-Block im RAM aktualisiert. Vergißt man das, kommt es bei einer anschließenden Leseoperation unweigerlich zu einer Fehlermeldung (BDOS Err on d:R/O). Bei einer Schreiboperation kann sogar ein Teil der gespeicherten Daten auf der Diskette zerstört werden. (Bei späteren CP/M-Versionen wurde dieses Problem abgefangen.)

Insgesamt gibt es die folgenden CTRL-Tasten-Kombinationen in CP/M (es ist also nötig, daß das angeschlossene Terminal bzw. die Tastatur diese auch erzeugen kann; auf dem Bildschirm erscheinen sie meist in Form des jeweiligen Buchstabens, dem ein Pfeil nach oben vorangestellt ist):

CTRL-C	CP/M-Warmstart durchführen
CTRL-E	Zeilenvorschub ausgeben, jedoch Kommandozeile nicht beenden
CTRL-H	Zuletzt eingegebenes Zeichen löschen, ein Zeichen zurück (Backspace; auch DEL = Rubout)
CTRL-J	Wirkt wie Return (Line Feed)
CTRL-M	Identisch mit Return: Zeile abschließen
CTRL-O	Eingegebene Zeile löschen und neue Zeile beginnen
CTRL-P	Protokollierung auf Drucker (LPT) ein- oder ausschalten
CTRL-R	Eingegebene Zeile nach Backspace wiederholen
CTRL-S	Anhalten einer Bildschirm-Ausgabe; nochmalige Eingabe läßt die Ausgabe weiterlaufen
CTRL-X	Kommandozeile löschen
CTRL-Z	Trennzeichen für Zeichenketten in der Kommandozeile

Bei der folgenden Kommando-Übersicht bezeichnet „d" jeweils das zu verwendende Disketten-Laufwerk (dafür sind die Buchstaben A...P zulässig, je nachdem, wie viele Laufwerke installiert sind). „Extern" bedeutet, daß das Kommando die entsprechende Datei vom Typ .COM von der CP/M-Systemdiskette laden muß, da die hierfür nötige Routine nicht im Betriebssystem-Kern enthalten ist. Vor den Dateinamen (hier wird NAME als Beispiel verwendet) darf auch eine Laufwerks-Bezeichnung stehen, so daß die Datei sich auf einem anderen als dem voreingestellten Laufwerk befinden kann, z.B. B:NAME.COM.

CP/M-Kommandos

d:
Definiert das voreingestellte Disk-Laufwerk um. Beispiel:
B:

ASM
8080-Assembler (extern). ASM assembliert die Quelltext-Datei NAME.ASM und erzeugt drei neue Dateien: NAME.PRN (Listing), NAME.SYM (Symbolta-

belle) und NAME.HEX (Maschinenprogramm). ASM wurde im Kapitel über Assembler bereits ausführlich beschrieben.

DDT
Da die meisten CP/M-Rechner kein Monitorprogramm im ROM besitzen, kann man mit DDT (extern) eines in den Arbeitsspeicher laden. Indem man DDT einen Dateinamen nachstellt, kann man eine Datei gleich hinzuladen, z.B. DDT NAME .COM. Unter DDT stehen folgende Kommandos zur Verfügung:

As	Mnemonische Befehlseingabe
D<s<,f>>	Speicherbereich s bis f anzeigen (hex und ASCII)
Fs,f,b	Speicherbereich s bis f mit dem Byte b füllen
G<s><bp>	Programm bei Adresse s starten, optional: bp = Breakpoint
Ha,b	Hexadezimale Summe und Differenz von a und b ausgeben
In	Name n für Kommando R setzen
L<s<,f>>	Speicherbereich s bis f disassemblieren
Ms,f,d	Speicherbereich s bis f nach d verschieben
R	Datei in den Speicher lesen (vgl. I)
Ss	Default-Adresse s vereinbaren (z.B. für D,G,L)
T<n>	Einzelschritt-Betrieb (optional: n Schritte)
U<n>	Programmablauf protokollieren (n Schritte)
X<r>	Prozessor-Register ändern

Optionale Angaben sind jeweils in spitze Klammern eingeschlossen. Nach dem Kommando-Buchstaben darf kein Leerraum stehen.

DCOPY oder DISKCOPY
Dieses Programm ist im Lieferumfang des CP/M von Digital Research nicht enthalten, wird jedoch von vielen Rechnerherstellern mitgeliefert. Es ermöglicht das Kopieren einer kompletten Diskette auf eine andere, sofern man über mindestens zwei Laufwerke verfügt.

DIR
Anzeigen des Inhaltsverzeichnisses einer Diskette (internes Kommando). SYS-Dateien (System-Dateien wie z.B. CP/M selbst) werden nicht angezeigt. Falls man Dateinamen nachstellt, kann man nicht relevante Zeichenfolgen durch einen Stern kennzeichnen (sog. Wildcard). Beispiele:

DIR	Gesamtes Verzeichnis des aktuellen Laufwerks
DIR NAME.*	Alle Dateien NAME mit beliebiger Extension anzeigen
DIR B:	Verzeichnis von Laufwerk B (gleichw. mit DIR B:*.*)
DIR B:*.ASM	Alle ASM-Dateien auf Laufwerk B anzeigen

DUMP
Ausgeben einer Datei in Form eines Hexadezimal-Speicherauszugs auf dem Bildschirm, nach CTRL-P auch auf den Drucker. Beispiel:
DUMP NAME.COM

ED
Zeileneditor (extern). Die Kommandos wurden bereits im Kapitel über Assembler besprochen. Ein Dateiname muß nachgestellt werden, z.B. ED NAME.$$$.

ERA
Löschen einer Datei (intern). Nicht relevante Zeichenfolgen können mit einem Sternchen markiert werden (Wildcard). Beispiele:

ERA NAME.EXT	Löscht die Datei NAME.EXT
ERA NAME.*	Lösche alle Dateien NAME mit beliebiger Extension
ERA *.EXT	Löscht alle Dateien mit der Extension EXT
ERA A*.ASM	Löscht alle ASM-Dateien, deren Name mit A beginnt

FORMAT
Dieses Programm (extern) wird zwar von Digital Research selbst nicht mit CP/M mitgeliefert, da CP/M zunächst für 8-Zoll-Disketten konzipiert war, die bereits formatiert erhältlich sind. Die Hersteller von Rechnern mit 5,25- oder 3,5-Zoll-Laufwerken liefern jedoch das FORMAT-Programm mit, damit der Benutzer fabrikneue Disketten selbst formatieren kann.

LOAD
Externes Kommando zum Erzeugen einer .COM- aus einer .HEX-Datei, z.B. um aus einem Assembler-Objektfile eine Kommandodatei zu machen. Der Teil des Namens vor dem Extension-Punkt wird beibehalten. Beispiel:

LOAD NAME.HEX (gleichwertig mit LOAD NAME)

MOVCPM
Konfiguriert eine CP/M-Version für eine bestimmte RAM-Größe in KByte (externes Kommando). Beispiel:

MOVCPM 60

PIP
Programm zum Datei-Transfer (extern), Wildcards (Sternchen) sind erlaubt. Folgendes Format ist notwendig (Q = Quelldatei, Z = Zieldatei, S = Sonderoption):

PIP Z=Q S

Hierbei können Z und Q entweder Dateinamen (ev. mit vorangestellter Laufwerksbezeichnung) oder Gerätenamen (mit nachgestelltem Doppelpunkt) sein. Als Sonderoptionen sind, in eckigen Klammern eingeschlossen (entsprechend Ä und Ü bei deutschem Zeichensatz!) folgende Zeichen oder Zeichenfolgen möglich:

A	Nur modifizierte Dateien übertragen
Dn	Alle Zeichen nach Spalte n ignorieren (Zeilenlänge!)
E	Echo zur Konsole
F	Formfeed-Zeichen (Seitenvorschub) nicht mitkopieren
G	Datei von User-Nummer n holen (vgl. Kommando USER)
H	Test auf korrektes Intel-Hex-Format

I	Alle mit :00 beginnenden Hex-Records einer Intel-Hex-Datei weglassen, Rest auf korrektes Format testen
L	Groß- in Kleinbuchstaben umsetzen (Lower Case)
N	Zeilennummern am Zeilenanfang ergänzen
O	Objektcode-Transfer; CTRL-Z nicht als Dateiende auffassen
Pn	Seitenlänge auf n Zeilen setzen (60 voreingestellt)
Qtext	Kopieren beenden, wenn die mit CTRL-Z abgeschlossene Zeichenfolge „text" erreicht ist
R	Dateien mit SYS-Attribut lesen
Stext	Kopieren ab der Zeichenfolge „text", die mit CTRL-Z enden muß
Tn	Tabulator auf n Leerzeichen setzen
U	Klein- in Großbuchstaben umsetzen (Upper Case)
V	Test auf korrekte Übertragung (Verify)
W	Schreibschutz ignorieren (Write Protected)
Z	Paritätsbit löschen (Bit 7 = 0)

Für die Geräteangaben sind folgende Namen möglich (sie wurden bereits weiter vorn erläutert): TTY, CRT, PTR, PTP, LPT, UC1, UR1, UR2, UP1, UP2, UL1. Alternativ können auch die logischen E/A-Kanäle angegeben werden (CON, LST, PUN, RDR, INP, OUT, PRN, EOF (generiert ein CTRL-Z) und NUL (generiert 40 ASCII-NUL-Zeichen)). Wird PIP ohne Datei-Angaben gestartet, meldet sich das Programm mit einem Stern als Prompt und erwartet dann die Parameter-Eingaben. Beispiele für die Anwendung:

PIP B:=A:NAME	Kopiert die Datei NAME von Laufwerk A nach B
PIP B:=A:*.*	Kopiert alle Dateien von Laufwerk A nach B
PIP NEU=ALT1,ALT2	Verkettet die Dateien ALT1 und ALT2 zu NEU
PIP LPT:=B:NAME	Gibt die Datei NAME von Laufwerk B auf den Drucker aus

REN
Umbenennen einer Datei (intern). Beispiele:

```
REN NEU=ALT
REN C:NEU=ALT
REN NEU=B:ALT
```

SAVE
Abspeichern eines Bereichs ab Adresse hex 0100 (d.h. im TPA-Bereich) auf Diskette. Die nach SAVE folgende Zahl gibt an, wie viele 256-Byte-Blöcke abzuspeichern sind. Beispiel:

```
SAVE 4 NAME.COM
```

STAT
Status von Laufwerken bzw. Disketten oder Peripheriegräten anzeigen (extern).

Beispiel:

STAT VAL:	Liste der möglichen STAT-Kommandos anzeigen
STAT	Freien Speicherplatz und Status einer Disk anzeigen: R/O = schreibgeschützt (Read Only), R/W = Read/Write
STAT DSK:	Laufwerk-Status anzeigen
STAT NAME	Länge der Datei NAME in KByte anzeigen
STAT USR:	Benutzte USER-Nummern anzeigen (siehe USER)
STAT B:=R/O	Laufwerk B schreibschützen
STAT NAME $R/O	Datei NAME schreibschützen
STAT NAME $R/W	Datei NAME zum Schreiben freigeben
STAT NAME $SYS	Datei NAME bei DIR nicht anzeigen
STAT DEV:	Zuordnung der logischen E/A-Kanäle zu physikalischen Peripheriegeräten anzeigen
STAT CON:=CRT:	Bildschirm-Terminal (CRT) als Konsole verwenden

SUBMIT
Startet die Ausführung einer Kommando-Datei (extern). Dabei wird Kommando für Kommando von einer .SUB-Datei in den Puffer geladen und ausgeführt, d.h. die Tastatureingabe wird sozusagen auf Disk umgeschaltet. Die Datei (max. 2048 Zeichen) darf pro Zeile (max. 125 Zeichen) nur ein Kommando enthalten. Beispiel:

SUBMIT NAME

SYSGEN
Externes Utility-Programm; kopiert das Betriebssystem von den Spuren 0 und 1 einer Quelldiskette auf eine andere Diskette, so daß diese auch gebootet werden kann (dies ist nach FORMAT noch nicht möglich, da sie ja noch kein Betriebssystem enthält!). Das Programm arbeitet dialogorientiert, es brauchen also keine Parameter zu folgen.

TYPE
Gibt eine Textdatei auf den Bildschirm aus (intern), nach CTRL-P auch auf den Drucker. Beispiele:

TYPE NAME.PRN
TYPE B:NAME.INF

USER
Unter den CP/M-Versionen ab 2.0 (also auch bei 2.2) kann man mit USER (intern) zwischen bis zu 16 verschiedenen Benutzer-Bereichen hin- und herschalten, wobei jeder sein eigenes Directory (DIR) hat. Voreingestellt ist der Bereich 0. Beispiel:

USER 2

XSUB
Erlaubt den Ablauf von Programmen, die mit gepufferter Eingabe arbeiten (z.B. ED), in einer SUBMIT-Datei und muß dann als erstes Kommando in dieser SUBMIT-Datei erscheinen.

Fehlermeldungen

?
Das eingegebene Kommando existiert nicht bzw. die zugehörige COM-Datei wurde nicht gefunden. Unter DDT: Falsche Mnemonic-Eingabe, Datei kann nicht eröffnet werden, falsche Prüfsumme einer Intel-Hex-Datei oder Assembler bzw. Disassembler wurde überschrieben. Bei USER: Unzulässige Benutzernummer (z.B. größer als 15).

??
DDT-Fehlermeldung beim Disassemblieren: Der gefundene Code ist kein 8080-Befehl (DDT beherrscht nicht die zusätzlichen Befehle des Z80!).

ABORTED
PIP wurde durch Drücken einer Taste abgebrochen.

BAD DELIMITER
Fehler in der Parameter-Liste nach STAT.

BAD LOAD
Fehlermeldung nach mißglücktem SAVE oder vom CCP.

BDOS ERROR ON d:

BAD SECTOR	Fehler beim Lesen von Disk. Return = Fehler ignorieren, CTRL-C = zurück zur CP/M-Kommandoebene
FILE R/O	Es wurde versucht, eine schreibgeschützte Datei zu ändern oder sie umzubenennen
R/O	Es kann nicht geschrieben werden, da entweder das Laufwerk mit STAT schreibgeschützt wurde oder die Diskette gewechselt wurde
SELECT	Das angegebene Laufwerk existiert nicht

Die BDOS-Fehlermeldungen warten alle auf einen Tastendruck. (Es ist daher z.B. in MBasic nicht möglich, sie mit ON ERROR abzufangen!)

BREAK x AT c
Meldung bei ED, wenn bei Kommando c ein Fehler auftrat. x enthält einen Fehlercode:

= Suchstring nicht gefunden
? Kommando existiert nicht
O Datei wurde nach R nicht gefunden
> String zu lang, Puffer voll
E Kommando durch Tastendruck abgebrochen
F Disk bzw. Directory voll

CANNOT CLOSE
Eine Datei kann nicht geschlossen werden.

CANNOT READ
PIP kann vom Lochstreifenleser nichts lesen.

CANNOT WRITE
PIP kann auf die Zieldatei nicht schreiben.

CHECKSUM ERR
Prüfsummen-Fehler bei einer HEX-Datei bei PIP.

COMMAND BUFFER OVERFLOW
Eine Submit-Datei ist länger als 2048 Zeichen.

COMMAND TOO LONG
Eine Submit-Zeile ist länger als 125 Zeichen.

CORRECT ERROR, TYPE RETURN OR CTRL-Z
Prüfsummen-Fehler einer HEX-Datei bei PIP.

DESTINATION IS R/O, DELETE (Y/N)?
Die Zieldatei bei PIP ist schreibgeschützt. Mit Y kann sie trotzdem beschrieben werden.

DIRECTORY FULL
Kein Platz mehr im Disk-Inhaltsverzeichnis. Bei ED kann man mit OXname eine nicht mehr benötigte Datei löschen, ohne daß der eingetippte Text verloren geht.

DISK FULL
Die Diskette ist voll, ED kann den Text nicht auf das angegebene Laufwerk schreiben.

DISK READ ERROR name
PIP fand ein CTRL-Z als Dateiende, bevor die Datei wirklich zu Ende war.

DISK WRITE ERROR name
Schreibfehler oder Diskette voll bei PIP, SUBMIT oder DDT.

ERROR: BAD PARAMETER
Die Parameter hinter PIP sind falsch angegeben.

ERROR: CANNOT OPEN SOURCE
LOAD kann die angegebene HEX-Datei nicht finden bzw. es wurde kein Name hinter LOAD spezifiziert.

ERROR: CANNOT CLOSE FILE
Die erzeugte COM-Datei kann von LOAD nicht geschlossen werden.

ERROR: DISK READ, LOAD ADDRESS xxxx
Bei der Hex-Adresse xxxx meldet LOAD einen Prüfsummen-Fehler der HEX-Datei.

ERROR: DISK WRITE, LOAD ADDRESS xxxx
Die erzeugte COM-Datei kann bei LOAD ab Adresse xxxx nicht weiter beschrieben werden (z.B. Diskette voll).

ERROR: INVERTED LOAD ADDRESS, LOAD ADDRESS xxxx
Es wurde versucht, eine Datei mit mehr als 32 KByte Länge mit LOAD vom HEX- in das COM-Format umzuwandeln (LOAD funktioniert nur bis 32 KByte!).

ERROR: NO MORE DIRECTORY SPACE, LOAD ADDRESS xxxx
Bei LOAD war es nicht möglich, die COM-Datei zu öffnen, weil das Disk-Inhaltsverzeichnis voll ist.

ERROR ON LINE xxx
Fehlermeldung von SUBMIT, wenn in Zeile xxx ein Kommando nicht decodiert werden konnte.

FILE ERROR
ED meldet damit, daß das Disk-Directory voll ist. Der eingetippte Text ist dann leider verloren (!).

FILE EXISTS
Es wurde versucht, eine Datei neu zu erzeugen, die bereits auf der Disk existiert (z.B. mit REN).

FILE EXISTS, ERASE IT
ED-Meldung beim Abspeichern einer Datei. Entweder muß man die alte Datei gleichen Namens löschen oder ein anderes Laufwerk verwenden.

FILE IS READ/ONLY
Es wurde versucht, von ED aus auf eine schreibgeschützte Datei zu schreiben.

FILE NOT FOUND
Fehlermeldung von STAT, ED oder PIP, wenn eine zu lesende Datei nicht auf der Diskette gefunden wurde.

FILENAME REQUIRED
Beim Aufruf von ED wurde der Dateiname weggelassen, was nicht zulässig ist.

INSUFFICIENT MEMORY
Es ist nicht genug Speicherplatz vorhanden, um von DDT aus die spezifizierte Datei zu laden.

INVALID ASSIGNMENT
Es wurde eine falsche Angabe bei der Zuweisung einer Datei oder eines Peripheriegeräts gemacht.

INVALID CONTROL CHARACTER
In einer SUBMIT-Datei befinden sich Steuerzeichen.

INVALID DIGIT name
PIP hat beim Lesen der HEX-Datei „name" ein Zeichen gefunden, das keine hexadezimale Ziffer ist.

INVALID DISK ASSIGNMENT
Unzulässige Angabe nach dem Gleichheitszeichen hinter der Laufwerksbezeichnung bei STAT.

INVALID DISK SELECT
Das spezifizierte Laufwerk existiert nicht.

INVALID DRIVE NAME
SYSGEN kann nur die Laufwerke A, B, C, D bedienen, andere führen zu dieser Fehlermeldung.

INVALID FILE INDICATOR
Fehlerhafte Angabe beim Setzen von Attributen für Dateien nach STAT.

INVALID FORMAT
Die hinter PIP stehenden Parameter sind so nicht verwertbar.

INVALID HEX DIGIT
Fehlermeldung von LOAD, wenn die HEX-Datei nach dem Doppelpunkt nicht hexadezimale Zeichen enthält.

INVALID MEMORY SIZE
Unzulässiger Parameter hinter MOVCPM.

INVALID SEPARATOR
Das Zeichen zwischen zwei Dateinamen muß bei PIP ein Gleichheitszeichen sein.

INVALID USER NUMBER
Fehlermeldung von PIP: CP/M 2.2 erlaubt nur USER-Nummern von 0 bis 15.

NO DIRECTORY SPACE
PIP findet im Disk-Inhaltsverzeichnis keinen Platz mehr für die zu erzeugende Datei.

NO FILE name
Die spezifizierte Datei wurde nicht gefunden (bei ASM, DDT, DIR, ERA, REN, PIP).

NO INPUT FILE PRESENT ON DISK
Die nach DUMP angegebene Datei wurde auf der Diskette nicht gefunden.

NO MEMORY
Es ist kein Puffer-Speicher mehr zum Laden einer Datei vorhanden.

NO SOURCE FILE ON DISK
Es wurde versucht, mit SYSGEN CP/M von einer Disk zu kopieren, die kein Betriebssystem auf den Systemspuren enthält.

NO SOURCE FILE PRESENT
ASM fand auf der Diskette keine passende Datei mit der Extension .ASM.

NO SPACE
Es ist kein Platz mehr für die von SAVE zu erzeugende Datei vorhanden.

NO SUB FILE PRESENT
Die im SUBMIT-Kommando angegebene Datei wurde nicht gefunden.

NOT A CHARACTER SOURCE
Für PIP wurde eine falsche Quelldatei (z.B. ein Drucker) spezifiziert.

NOT DELETED
Meldung nach DESTINATION IS R/O, DELETE (N/Y), wenn nicht Y eingegeben wurde: Die schreibgeschützte Datei wurde von PIP nicht überschrieben.

NOT FOUND
PIP konnte die angegebene Datei auf der Diskette nicht finden.

OUTPUT FILE WRITE ERROR
Unter ASM trat beim Schreiben einer Datei ein Fehler auf.

PARAMETER ERROR
Die Parameter einer SUB-Datei müssen die Namen $1 bis $9 haben.

PARAMETER ERROR, TYPE RETURN TO IGNORE
Durch Drücken von Return kann bei SYSGEN ein falsch angegebener Parameter ignoriert werden.

QUIT NOT FOUND
Die als Endkennzeichen bei der PIP-Option Q definierte Zeichenfolge wurde bis zum Dateiende nicht gefunden.

READ ERROR
Disk-Lesefehler beim Ausgeben einer Textdatei mit TYPE.

READER STOPPING
Bei PIP kommen vom Lochstreifenleser keine Zeichen mehr, bevor die Datei zu Ende ist.

RECORD TOO LONG
Die maximale Record-Länge bei PIP beträgt 128 Byte.

REQUIRES CP/M 2.0 OR LATER
Es wird versucht, XSUB mit einer CP/M-Version vor 2.0 zu verwenden.

REQUIRES CP/M 2.0 OR NEWER FOR OPERATION
Die PIP-Version von CP/M 2.2 kann nicht mit einer früheren CP/M-Version zusammenarbeiten.

START NOT FOUND
Die bei PIP mit der Option S als Start angegebene Zeichenfolge wurde bis zum Dateiende nicht gefunden.

SOURCE FILE INCOMPLETE
ASM meldet, daß die soeben gelesene Assembler-Source-Datei offensichtlich nicht vollständig ist (z.B. fehlendes CTRL-Z am Ende).

SOURCE FILE NAME ERROR
Hinter ASM darf man in einem Dateinamen kein Wildcard-Zeichen (* oder ?) verwenden.

SOURCE FILE READ ERROR
Beim Lesen der ASM-Quelltext-Datei trat ein Fehler auf.

SYNCHRONIZATION ERROR
Die Versionsnummern von MOVCPM und dem Betriebssystem selbst stimmen nicht überein.

SYSTEM FILE NOT ACCESSIBLE
Mit STAT lassen sich keine System-Dateien ansprechen.

TOO MANY FILES
STAT-Fehlermeldung; die spezifizierten Dateien können nicht innerhalb eines Befehls bearbeitet werden.

UNEXPECTED END OF HEX FILE name
PIP hat beim Kopieren einer HEX-Datei deren letzte Zeile (beginnt mit :00) nicht gefunden.

UNRECOGNIZED DESTINATION
Die Angabe der Zieldatei wurde von PIP nicht verstanden.

VERIFY ERROR name
PIP hat mit der Option V (Verify) einen Fehler beim Kopieren festgestellt.

WRONG CP/M VERSION (REQUIRES 2.0)
Die verwendete CP/M-Version muß für das spezifizierte Kommando mindestens 2.0 sein.

XSUB ACTIVE
XSUB wird gerade ausgeführt, keine Tastatureingabe möglich.

XSUB ALREADY PRESENT
XSUB wurde unnötigerweise aufgerufen, obwohl es bereits im Speicher steht.

Möglicherweise ist es frustrierend, an Hand dieser zahlreichen Fehlermeldungen zu sehen, was man alles verkehrt machen kann. Noch viel frustrierender ist es bei CP/M 2.2 allerdings, daß gerade die BDOS-Fehler, die (wenn ein Anwender-Programm erst einmal fertig entwickelt ist und läuft) am wahrscheinlichsten auftreten, nicht abgefangen werden können (z.B. mit ON ERROR GOTO in MBasic), d.h. der nicht vorgebildete Benutzer erhält eine für ihn vermutlich unverständliche englische Systemmeldung. Dieser erhebliche Mangel ist bei der (allerdings speicher-

platz-fressenden) Version CP/M 3.0 behoben worden — ebenso das Problem, daß man bei CP/M 2.2 nach jedem Diskettenwechsel erst CTRL-C (bzw. in MBasic RESET) eingeben mußte, um nicht durch einen BDOS-R/O-Fehler „abzustürzen" und womöglich mühsam eingegebene Daten unrettbar zu verlieren.

BDOS-Funktionen

Damit man Files nicht nur vom CP/M-CCP aus, sondern auch z.B. unter Basic oder Pascal laden und abspeichern kann, schreibt CP/M in die Adresse 0005 einen Sprung zum BDOS, so daß man mit der CALL-Anweisung des 8080 bzw. Z80 das BDOS als Unterprogramm aufrufen kann. Welche BDOS-Funktion dabei ausgeführt wird, hängt vom Inhalt des C-Registers der CPU ab: Jede Funktion besitzt eine Nummer, die vor dem Befehl CALL 0005 in das C-Register zu laden ist. In der folgenden Aufstellung ist zuerst diese Nummer und dann die Wirkung der zugehörigen Funktion genannt.

0 CP/M-Warmstart (Sprung zum CCP ohne Return)
1 Konsolen-Eingabe, Zeichen in A
2 Konsolen-Ausgabe, Zeichen in E
3 Reader-Eingabe, Zeichen in A
4 Punch-Ausgabe, Zeichen in E
5 List-Ausgabe, Zeichen in E
6 E = FF: Konsol-Eingabe, Zeichen in A
E = Zeichen: Konsol-Ausgabe, Status in A
7 I/O-Byte ins A-Register holen
8 E-Register ins I/O-Byte schreiben
9 String ab der Adresse in DE drucken, Ende = $
10 Eine Zeile in den Konsolen-Puffer einlesen; DE = Adresse
11 Konsolen-Status in A-Register liefern
12 Liefert CP/M-Versionsnummer in HL
13 Reset-Funktion für alle Laufwerke (vgl. RESET in MBasic!)
14 Laufwerk einstellen, E = Nummer (0 = Laufwerk A)
15 Datei öffnen, DE = FCB-Adresse; A = FF: nicht gefunden
16 Datei schließen, DE = FCB-Adresse; A = FF: nicht gefunden
17 Datei im Inhaltsverzeichnis suchen, DE = FCB-Adresse, A = FF: nicht gefunden
18 Im Inhaltsverzeichnis weiter suchen, DE = FCB-Adresse, A = FF: nicht gefunden
19 Datei löschen, DE = FCB-Adresse; A = FF: nicht gefunden
20 Datei sequentiell lesen, DE = FCB-Adresse, A = 0: kein Fehler
21 Datei sequentiell schreiben, DE = FCB-Adresse; A = 0: kein Fehler
22 Neue Datei öffnen, DE = FCB-Adresse; A = FF: Diskette voll
23 Datei umbenennen, DE = FCB-Adresse; A = FF: nicht gefunden
24 HL mit Bitmuster laden, das bisher angesprochene Laufwerke enthält (Bit 0 von L = Laufwerk A)

25 Aktuelles Laufwerk ermitteln, Nummer in C (0...15)
26 DMA-Adresse setzen (steht in DE)
27 Adresse des Bitmusters freier Sektoren in HL laden
28 Aktuelles Laufwerk schreibschützen
29 Bitmuster der schreibgeschützten Laufwerke in HL laden
30 Datei-Attribute setzen, DE = FCB-Adresse; A = FF: nicht gefunden
31 Adresse des Disk-Parameter-Blocks (DPB) in HL laden
32 USER-Nummer holen (E = FF) oder setzen E = Nummer 0...15)
33 Random-Lesezugriff, DE = FCB-Adresse; A = Fehlercode: 0 = OK
34 Random-Schreibzugriff, DE = FCB-Adresse; A = Fehlercode: 0 = OK
35 Dateilänge errechnen, DE = FCB-Adresse; liefert Blockzahl im FCB
36 Random-Feld im FCB (Adresse = DE) setzen
37 Laufwerk(e) zurücksetzen, DE = Bitmuster der Laufwerke (Bit 0 in Reg. E = Laufwerk A)

CP/M-Plus (CP/M 3.X)

Einige Schwächen von CP/M 2.2 wurden mit der Version 3.0 behoben, die auch CP/M-Plus genannt wird. Es gibt sie in zwei Versionen — eine für 64-KByte-Systeme ohne Bank-Umschaltung, die andere für Systeme mit mehr als 64 KByte RAM und einer RAM-Bank-Umschaltlogik.

Die Speicherverwaltung der „ungebankten" Version ähnelt der von CP/M 2.2. Allerdings verbleibt weniger freier Speicherplatz für Anwenderprogramme (TPA), obwohl weniger Funktionen als beim „gebankten" CP/M-Plus zur Verfügung stehen (kein Paßwort-Schutz, weniger Editiermöglichkeiten). Gegenüber CP/M 2.2 sind als wesentliche Verbesserungen zu nennen: BDOS-Fehlermeldungen können abgefangen werden und sind auch ausführlicher; nach dem Wechseln einer Diskette ist es nicht nötig, CTRL-C zu drücken bzw. die BDOS-Reset-Funktion aufzurufen; der Zugriff auf das Inhaltsverzeichnis der Disk erfolgt durch Zwischenspeicherung im RAM schneller; an Massenspeicher können bis zu 512 MByte verwaltet werden (CP/M 2.2: 8 MByte).

Weitere Vorteile von CP/M-Plus sind: automatisches Führen von Kalender und Uhrzeit im Disk-Directory; RSX-Moduln (Resident System Extension) zur Schaffung neuer, im Speicher verbleibender Kommandos; einfachere Zuordnung von I/O-Kanälen (Beispiel: GET CONSOLE INPUT FROM FILE TEST; PUT CONSOLE OUTPUT TO FILE TEST2); neues ARCHIVE-Attribut, um von wichtigen Dateien automatisch eine Sicherheitskopie anzufertigen. Bei der gebankten Version steht ein Paßwort-Schutz für einzelne Dateien in mehreren Stufen zur Verfügung: Löschsperre, Änderungssperre, Zugangssperre (jeweils nur mit Paßwort aufhebbar).

Die Speicheraufteilung der CP/M-Plus-Version mit Bankumschaltung zeigt *Abb. 4.1.1.2.* Insgesamt sind bis zu 16 „Bänke" möglich, wobei jedoch stets nur der

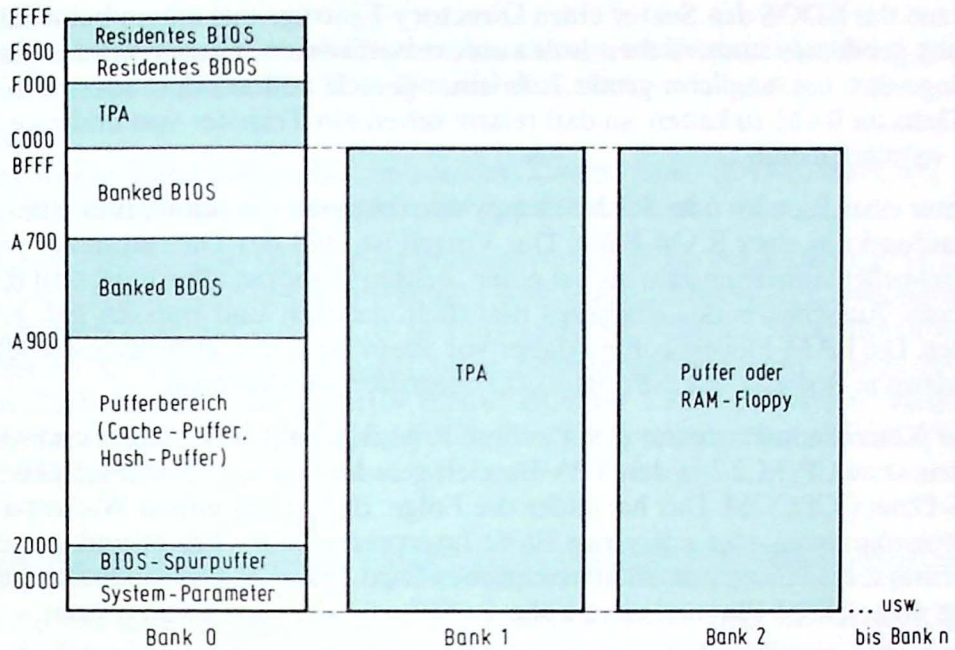


Abb. 4.1.1.2 Speicherverwaltung durch das Betriebssystem CP/M-3 (CP/M-Plus)

Bereich 0000...BFFF umgeschaltet wird; die obersten 16 KByte von Bank 0 sind auch bei Umschaltung auf eine andere Bank als gemeinsamer Bereich verfügbar. Dies ist nötig, weil z.B. die Routinen für die Bankumschaltung ja von jeder Bank aus ansprechbar sein müssen. Im gemeinsamen Bereich befindet sich auch ein Teil der TPA (C000...EFFF). Von Bank 0 wird der größte Teil für CP/M-Plus selbst und für seine Puffer verwendet, während Bank 1 (48 KByte) ausschließlich für Anwenderprogramme dient (TPA). Insgesamt stehen in Bank 0 und 1 also 60 KByte an freiem Speicher für Anwenderprogramme zur Verfügung.

Am Rande sei bemerkt, daß bei der CP/M-Plus-Konfiguration immerhin 16 KByte eines 128-KByte-Rechners völlig brach liegen, wenn der Speicher in zwei 64-KByte-Hälften organisiert ist, da auf die obersten 16 KByte ja immer von Bank 0 aus zugegriffen wird. Die „verschwendeten" 16 KByte könnte man zwar durch eine zusätzliche Banking-Logik zugänglich machen, von CP/M-Plus selbst wird dies jedoch nicht unterstützt. Stehen mehr als 128 KByte RAM zur Verfügung, so verwendet man die Bänke ab Nr. 2 entweder als zusätzliche Puffer für CP/M oder als Pseudo-Floppy.

Die zusätzlichen Puffer werden unter CP/M-Plus als sogenanntes Cache-Memory eingesetzt. So nennt man hier einen Speicherbereich, in den sozusagen „auf Verdacht" das Disk-Inhaltsverzeichnis gelesen wird, so daß es ohne weiteren (zeitraubenden) Disk-Zugriff sofort zur Verfügung steht, wenn z.B. Spur und Sektor einer bestimmten Datei, die gelesen werden soll, ermittelt werden müssen. Ein anderer Verwendungszweck für die CP/M-Plus-Puffer sind „Hashing-Tabellen": Da-

mit kann das BDOS den Sektor eines Directory-Eintrags aus einem beim Schreibvorgang gebildeten numerischen Index sofort bestimmen. Ferner werden die Puffer eingesetzt, um möglichst große Teile einer gerade zu lesenden oder zu ändernden Datei im RAM zu halten, so daß relativ selten ein Transfer von und zum Laufwerk stattfinden muß.

Unter einer Pseudo- oder RAM-Floppy versteht man die Simulation eines Floppy-Laufwerks in einer RAM-Bank. Der Vorteil ist, daß der Datentransfer wesentlicht schneller vonstatten geht als bei einer „echten" Floppy; allerdings sind die Daten beim Ausschalten des Rechners natürlich verloren und müssen ggf. gerettet werden. Die RAM-Floppy kommt daher vor allem für nur vorübergehend benötigte Dateien in Betracht, auf die relativ oft zugegriffen werden muß.

Der Kommando-Prozessor (CCP) von CP/M 3.X ist 4 KByte lang und wird im Gegensatz zu CP/M 2.2 in den TPA-Bereich geladen; auf der Systemdiskette steht er als Datei CCP.COM. Das hat leider die Folge, daß er bei einem Warmstart Anwenderprogramme oder z.B. einen Basic-Interpreter überschreibt und umgekehrt selbst von diesen Programmen überschrieben wird, was z.B. ein wechselweises Arbeiten unter CP/M-Plus und einer höheren Sprache sehr erschweren kann.

CP/M-Plus besitzt gegenüber CP/M 2.2 eine Reihe neuer (externer) Kommandos; dafür gibt es z.B. den STAT-Befehl hier nicht mehr:

HELP
Gibt einen Überblick über die 24 Kommandos von CP/M-Plus, ohne jedesmal im Handbuch nachschlagen zu müssen. Dem Wort HELP kann ein Befehl nachgestellt werden, über den man nähere Informationen wünscht, z.B. HELP DEVICE.

DEVICE
Mit DEVICE kann man Baudraten und Protokolle der 16 physikalischen Ein- und Ausgabegeräte festlegen sowie deren Zuordnung zu den logischen Kanälen CONSOLE, LIST und AUXILIARY umdefinieren. Beispiel:

```
CONIN:=CRT
CONOUT:=CRT
```

DIR
Unter CP/M-Plus ist auch DIR ein externes Kommando und gestattet u.a. auch die alphabetische Anzeige der Dateinamen oder der Dateien unterschiedlicher USER-Bereiche.

GET und PUT
Das Kommando GET schaltet die Konsolen-Eingabe auf eine Disketten-Datei um. Umgekehrt kann man mit PUT die Konsolen-Ausgabe auf eine Disk-Datei umleiten.

SET
Dient zum Setzen von Dateiattributen. Unter CP/M 2.2 mußte man hierzu entsprechende Kommandos nach STAT verwenden.

SETDEF
Hiermit ist es möglich, CP/M-Plus zu veranlassen, mehr als ein Laufwerk nach einer bestimmten COM- oder SUB-Datei abzusuchen.

SHOW
Dieses Kommando entspricht in seinem Zweck dem STAT-Kommando unter CP/M 2.2, liefert jedoch wesentlich ausführlichere Daten.

Auch die BIOS- und BDOS-Funktionsaufrufe wurden gegenüber CP/M 2.2 erweitert. Trotzdem sind beide Versionen weitgehend miteinander kompatibel, jedenfalls kann man mit einem CP/M-Plus-System CP/M-2.2-Disketten (trotz des erweiterten CP/M-Plus-Directory) ohne Schwierigkeiten lesen. Auch der von CP/M 2.2 bekannte „File Control Block" ist unter CP/M 3.X ebenso aufgebaut, läßt sich jedoch für die speziellen neuen Disk-Operationen zum „XFCB" (Extended FCB) erweitern.

MP/M

Auch MP/M stammt von Digital Research und wird hier innerhalb dieses Kapitels mit aufgeführt, weil es sehr große Ähnlichkeiten mit CP/M aufweist. MP/M ist ein Betriebssystem für bis zu acht Benutzer, die sich einen Rechner im Timesharing-Betrieb teilen.

Als Prompt meldet sich MP/M auf jedem der angeschlossenen Terminals mit der jeweiligen Benutzernummer und dem aktuellen Disketten- bzw. Plattenlaufwerk, z. B.

3A>

Natürlich darf ein Benutzer nicht einfach eine Disk in einem Laufwerk austauschen, da dann die anderen Benutzer womöglich auf „ihre" Dateien nicht mehr zugreifen können. Will man also eine Diskette wechseln, so gibt man auf einem der Terminals ein:

DISKRESET

Auf allen angeschlossenen Terminals fragt der MP/M-Rechner dann: Confirm Disk Reset System (Y/N)? Wenn ein oder mehrere Benutzer mit N antworten, wird der Befehl zurückgewiesen. Weitere typische MP/M-Befehle sind:

CTRL-D	Das im Vordergrund von der Konsole gestartete Programm wird auf Hintergrund-Betrieb umgeschaltet
ABORT	Das laufende Programm wird abgebrochen
ATTACH	Ein im Hintergrund laufendes Programm wird in den Vordergrund zurückgeholt
CONSOLE	Erfragen der von MP/M zugeteilten logischen Konsolen-Nummer (nicht mit Benutzer-Nummer identisch!)
SCHED	Aufstellen eines Zeitplans, in welcher Reihenfolge bestimmte Programme gestartet werden sollen

SPOOL	Übergabe einer Datei an einen Zwischenpuffer, der sie auffängt, bis z. B. der Drucker frei ist
TOD	Abfragen und Setzen von Uhrzeit und Datum

Der Befehl SPOOL ist bei Mehrbenutzer-Systemen immer dann notwendig, wenn die Zahl der angeschlossenen Drucker geringer als die der Benutzer ist. Während ein Spooler bei Einbenutzer-Rechnern die zu druckenden Zeichen im Arbeitsspeicher oder einem besonderen Spooler-RAM puffert, sorgt der SPOOL-Befehl bei MP/M dafür, daß die zu druckende Datei von der Diskette oder dem Festplatten-Laufwerk geholt und gedruckt wird, sobald der Drucker wieder verfügbar ist. Dabei können auch mehrere Druckaufträge unterschiedlicher Benutzer „gestapelt" werden.

Damit die einzelnen Benutzer mit unterschiedlichen Programmen (Tasks) arbeiten können, enthält MP/M einen besonderen Betriebssystemteil namens XDOS, der sich darum kümmert, daß nacheinander zyklisch alle Tasks abgearbeitet werden.

CP/M-86 und Concurrent-CP/M-86

Die bisher besprochenen CP/M-Versionen sind ausschließlich auf Rechnern lauffähig, die eine der CPU-Typen 8080, 8085, Z80 ohne NSC-800 verwenden. Unter dem Namen CP/M-86 hat Digital Research eine Version auf den Markt gebracht, die auf den Befehlssatz der Prozessoren 8088 und 8086 bzw. kompatible Nachfolger wie 80186 und 80286 zugeschnitten ist. Das Disketten-Aufzeichnungsformat ist mit CP/M-80 kompatibel, so daß zumindest ein Transfer von Source- bzw. Textdateien mit Z80- bzw. 8080-Rechnern ohne weiteres möglich ist. (8080-Maschinenprogramme kann die CPU 8086 ja nicht ausführen.)

Die Kommandos und Eigenschaften von CP/M-86 entsprechen im wesentlichen denen von CP/M 2.2. (Eine Version namens CP/M-86-Plus war bei Drucklegung dieses Buches bei Digital Research in Entwicklung, jedoch noch nicht auf dem Markt verfügbar, womit die von CP/M 2.2 übernommenen Schwächen zum großen Teil beseitigt sein dürften.) Im Gegensatz zu dem etwas verbreiteteren 8086-Betriebssystem MS-DOS von Microsoft wird bei CP/M-86 ein 8086-Assembler mitgeliefert.

Um CP/M-86 herum hat Digital Research eine Reihe von passenden Software-Paketen verfügbar gemacht, so etwa ein Grafik-Paket namens GSX sowie Compiler für höhere Programmiersprachen wie CBasic, Pascal MT-Plus, CIS-Cobol und andere (sie sind allerdings auch für das Betriebssystem MS-DOS verfügbar).

Aus CP/M-86 wurde von Digital Research auch Concurrent-CP/M-86 (CCPM) entwickelt. Dabei handelt es sich um ein Multitasking-Betriebssystem, unter dem bis zu vier Prozesse (Tasks) quasi gleichzeitig im Zeitmultiplex-Betrieb laufen können. Natürlich ist es auch möglich, jeder Task einen Benutzer zuzuordnen, so daß CCPM auch als Multiuser-System mit bis zu vier angeschlossenen Ter-

minals arbeiten kann. Die Betriebssystem-Teile XIOS.SYS (vgl. MP/M!), CCPM.SYS und BDOSPC.CON, die gleichzeitig im Speicher sein müssen, belegen zusammen allerdings allein schon 140 KByte, so daß man mit RAM in einem solchen System nicht geizen sollte.

Eine ganz besondere Eigenschaft von CCPM stellt die Fähigkeit dar, das „Konkurrenz-Betriebssystem" MS-DOS von Microsoft (siehe nächstes Kapitel) emulieren zu können. Das bedeutet, daß man Software, die ursprünglich für MS-DOS geschrieben wurde, unter CCPM laufen lassen kann, ohne das MS-DOS-System kaufen zu müssen. CCPM stellt durch Analyse der Directory-Spuren einer Diskette selbst fest, ob es sich um eine MS-DOS- oder CP/M-Diskette handelt. Erstaunlicherweise ist CCPM (V. 3.1) im MS-DOS-Emulationsmodus beim Schreiben auf Diskette sogar um etwa ein Drittel schneller als MS-DOS 1.0 selbst (nicht jedoch bei Verwendung einer RAM-Floppy oder Festplatte). Die CCPM-Version 3.1 beherrschte allerdings noch nicht das mit MS-DOS 2.0 eingeführte 9-Sektor-Aufzeichnungsformat (MS-DOS 1.0 und CCPM 3.1: 8 Sektoren pro Spur) sowie die sogenannten Sub-Directories (Unter-Inhaltsverzeichnisse).

Darüber hinaus unterstützt Concurrent-CP/M-86 auch die Unterteilung des Bildschirms in mehrere Arbeitsbereiche, wie das bereits im Kapitel über GW-Basic besprochen wurde (Windows = Fenster).

CP/M-68k

Auch für den von Motorola entwickelten 16-Bit-Prozessor 68000 gibt es eine CP/M-Version. CP/M-68k wurde von Digital Research in der Sprache C geschrieben. Den C-Sourcecode erhält man allerdings nicht mitgeliefert, nur den Objektcode zusammen mit einem für die Ausführung nötigen Laufzeit-Paket (Run Time Pakkage, RTP). Die zur Verfügung stehenden Kommandos entsprechen weitgehend CP/M-80, Version 2.2. Der CP/M-68k-Kern besteht aus BDOS, RTP, CCP und BIOS; wegen seines Umfangs ist ein Arbeitsspeicher des 68000-Rechners von mindestens 128 KByte unbedingte Voraussetzung.

Auf der Diskette befinden sich neben dem Betriebssystem-Kern selbst ein Zeileneditor (ED-68K), ein 68000-Assembler (ASM-68K) und ein passendes Programm zum Drucken der Assembler-Symboltabelle (NMC-68K), ein Linker/Loader zum Verbinden und Laden des vom Assembler erzeugten relokatiblen Objektcodes (LO-68K), eine Makro-Source-Bibliothek (AR-68K), ein Monitorprogramm bzw. Debugger (DDT-68K), das ebenfalls von CP/M-80 bekannte PIP-Programm zum Dateitransfer (PIP-68K), ein Hilfsprogramm zur Batch-Verarbeitung (SUBMIT), ein Programm zur Umwandlung eines Objektprogramms in das Motorola-S-Format sowie STAT, um Disk-Parameter abzufragen bzw. zu setzen (STAT-68K).

Im Gegensatz zu CP/M-80 wird CP/M-68k nicht unbedingt an die höchste zur Verfügung stehende Speicher-Adresse geladen; vielmehr kann der Benutzer die

Lage im Speicher selbst wählen. Auf den Disk-Systemspuren befindet sich ferner nicht das Betriebssystem CPM.SYS selbst, sondern lediglich ein Urlader. Direkt ausführbare Dateien besitzen hier nicht die Extension .COM, sondern .REL, da 68000-Programme normalerweise relokatibel (im Speicher verschiebbar) sind.

Zur Anzeige von SYS-Dateien steht ein eigenes Kommando zur Verfügung: DIRS. Wenn man nach TYPE keinen Dateinamen schreibt, fragt CP/M-68k automatisch danach, ebenso bei ERA und REN. Will man eine Ausgabe anhalten und dann weiterlaufen lassen, so geschah das bei CP/M-2.2 jeweils mit CTRL-S; CP/M-68k hält sich dagegen an das XON-XOFF-Protokoll und verwendet CTRL-S und CTRL-Q. Schließlich gibt es bei CP/M-68k kein SUBMIT; dies ist auch nicht mehr erforderlich, weil der Aufruf einer Datei mit der Extension .SUB nun die gleiche Wirkung hat. BDOS-Funktionen werden durch einen Software-Interrupt mit der Funktionsnummer im CPU-Register D0 und ggf. einem Parameter in D1 aufgerufen; Resultate werden wieder in D0 übergeben.

4.1.2 MS-DOS

Noch bevor CP/M von Digital Research auf die Prozessoren der 8086-Familie übertragen wurde, versuchte eine kleine Firma im Westen der USA, für einen von ihr auf der Basis des S100-Bussystems gebauten 8088-Computer CP/M auf der neuen CPU nachzuempfinden: Seattle Computer Products entwickelte das CP/M-ähnliche 8086/8088-Betriebssystem QDOS, das 1980 lieferbar war und damals bereits viele Eigenschaften von CP/M-Plus vorwegnahm. Leider war QDOS kein Verkaufserfolg, es fehlte am Marketing. Daran änderte sich auch nichts, als 1981 eine verbesserte Version von QDOS herauskam — bis sich Microsoft, von IBM nach einem passenden Betriebssystem für den gerade in Entwicklung befindlichen ersten IBM-PC befragt, entschloß, Mitte 1981 QDOS von Seattle Computer Products zu erwerben. Ende 1981, also wenig später, ließ IBM den PC aus dem Sack und nannte Microsofts MS-DOS einfach PC-DOS.

Dieser Tatsache ist wohl zu verdanken, daß MS-DOS bei Tischcomputern, die mit den Prozessoren 8088, 8086, 80186 oder 80286 arbeiten, wesentlich höhere Marktanteile erringen konnte als das Konkurrenz-Betriebssystem CP/M-86. Zahlreiche Firmen (darunter auch so angesehene wie Texas Instruments, Olivetti oder Siemens, aber auch vorher unbekannte) kamen bis Mitte der achtziger Jahre mit MS-DOS-Rechnern heraus. MS-DOS wurde, ob es der Benutzer wollte oder nicht, fast automatisch beim Kauf des Rechners mitgeliefert. Und in der Tat besitzt es zahlreiche Vorteile gegenüber dem betagten CP/M. Allerdings ist es auch sehr stark an die Hardware-Struktur des IBM-PC gebunden, was die große Ähnlichkeit der MS-DOS-Rechner untereinander erklärt. Im folgenden wird die Version 2.1 angenommen.

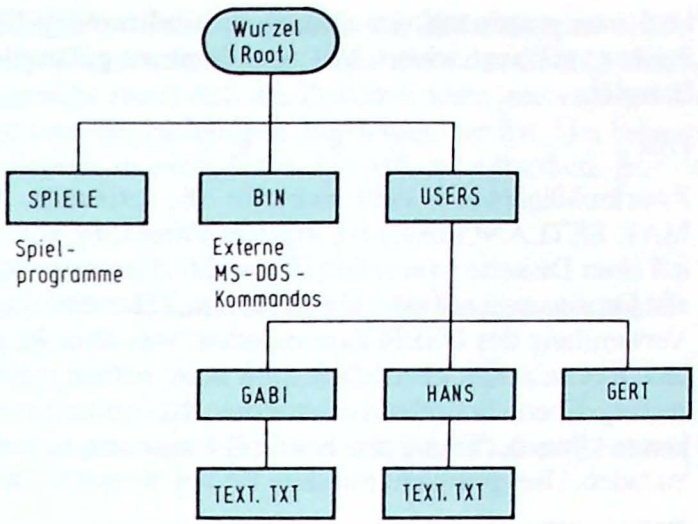


Abb. 4.1.2.1
MS-DOS gestattet wie Unix die Unterteilung von Disketten-Inhaltsverzeichnissen (Directories) in einer baumartigen Struktur

Directory-Struktur

Eine der herausragenden Eigenschaften von MS-DOS 2.0 und späterer Versionen ist die hierarchische Struktur des Disk-Inhaltsverzeichnisses. Genauer gesagt, es gibt mehrere Inhaltsverzeichnisse, die in einer baumähnlichen Struktur verkettet sind. Allerdings kann man ebensogut mit einem einzigen, herkömmlichen aufgebauten Directory wie z. B. unter CP/M arbeiten.

Abb. 4.1.2.1 zeigt ein Beispiel für einen Directory-Baum. Von der „Wurzel" aus (oben) kann man mit dem Kommando MKDIR (make Directory) Subdirectories eröffnen, im Beispiel eines, das SPIELE heißt und alle Spielprogramme der Disk aufführt, eines namens BIN für die wie bei CP/M erforderlichen externen Kommandos (deren Routinen nicht im Betriebssystem-Kern resident sind, sondern die bei Bedarf geladen werden müssen), ferner ein Subdirectory namens USERS für drei Benutzer, die den Computer regelmäßig bedienen. Es ist ohne weiteres möglich, daß zwei Dateien, die nicht im gleichen Subdirectory stehen, denselben Namen haben (hier TEXT.TXT), da der Zugriff zu ihnen ja nur über einen bestimmten, eindeutigen „Pfad" (path) stattfinden kann.

Wenn man in MS-DOS einen Dateinamen angibt, so ist ihm deshalb gegebenenfalls der Pfad durch die Subdirectories voranzustellen, damit der Rechner ihn finden kann. Dies kann in unserem Beispiel z. B. so geschehen:

```
TYPE A:\USERS\HANS\TEXT.TXT
```

(Der nach links geneigte Schrägstrich wird englisch auch „Backslash" genannt.) Man könnte sich aber auch zunächst auf die Directory-Ebene HANS begeben:

```
CD\USERS\HANS
```

Daß man gerade auf der aktuellen Subdirectory-Ebene arbeitet, wird mit einem Punkt (.) gekennzeichnet. Mit zwei Punkten gelangt man um eine Stufe höher, zum Beispiel:

DIR ..

Zweckmäßigerweise sieht man für die externen MS-DOS-Kommandos (FORMAT, SETLANG usw.) ein eigenes Directory vor. Damit lassen sich die Dateien auf einer Diskette wesentlich übersichtlicher verwalten als bei CP/M, wo nach DIR alle Dateinamen auf einmal erscheinen. (Der einzige Ausweg unter CP/M wäre die Verwendung des USER-Kommandos, was aber kein echter Ersatz ist, weil es nur eine Ebene zuläßt.) Natürlich muß man, sofern man sich auf einer anderen Subdirectory-Ebene befindet, bei externen Kommandos den Rechner irgendwie veranlassen können, die entsprechende Routine von einem ganz anderen Directory-Pfad zu laden. Dies geschieht mit dem PATH-Kommando, z. B.:

PATH \BIN

Diese hierarchische Pfad-Struktur wird auch von GW-Basic und BASICA (IBM-PC) unterstützt. Diese Basic-Interpreter sind deshalb (im Gegensatz z. B. zu MBasic-86) nur unter MS-DOS lauffähig, nicht z. B. unter CP/M-86.

Batch-Files, I/O-Umleitung, Filter

Eine weitere Besonderheit von MS-DOS ist, daß das System beim Booten automatisch nach einer Datei namens AUTOEXEC.BAT sucht, die (wie bei SUBMIT unter CP/M) MS-DOS-Kommandos enthalten kann, die so beim Einschalten des Rechners automatisch durchgeführt werden. Das ist z. B. nützlich, um mit SETLANG den richtigen nationalen Zeichensatz einzustellen oder Anwenderprogramme automatisch zu starten. Die Datei AUTOEXEC.BAT kann z. B. mit dem Texteditor EDLIN erstellt werden. Sie darf (wie andere Batch-Files auch) sogar Dummy-Zeichen enthalten. Im Beispiel bestehe die Batch-Datei namens TEST.BAT aus folgenden Zeilen:

1. COPY %1.TXT %2.TXT
2. TYPE %2.PRN
3. TYPE %0.BAT

Die unter MS-DOS eingetippte Anweisungsfolge

TEST A:PROG1 B:PROG2

ruft dann zunächst das Batch-File TEST.BAT auf und ersetzt %1 durch PROG1, %2 durch PROG2 und %0 durch TEST. Die obige Anweisung wäre also identisch mit:

COPY A:PROG1.TXT B:PROG2.TXT
TYPE B:PROG2.PRN
TYPE TEST.BAT

Schließlich ist es noch möglich, Ein- und Ausgaben vom Bildschirm auf ein beliebiges Gerät bzw. auch auf eine Datei umzuleiten. Dazu dient das Größer-Zeichen. Mit zwei solchen Zeichen erreicht man, daß die Zieldatei nicht neu eröffnet wird, sondern daß die neuen Daten an die bisherigen angehängt werden. Um beispielsweise das Disk-Inhaltsverzeichnis in eine Datei NAME zu schreiben, gibt man ein:

DIR>NAME

Um die Datei NAME um das Disk-Inhaltsverzeichnis zu verlängern, ohne sie zu löschen, lautet das Kommando:

DIR>>NAME

Die Umleitung von Tastatur-Eingaben ist äquivalent mit dem Kleiner-Zeichen (<) möglich.

Sogenannte Filter ermöglichen es in MS-DOS (wie bei Unix oder OS-9!), eine Datei zu lesen, zu ändern und in eine andere Datei zu schreiben: FIND sucht eine bestimmte Zeichenfolge in der Quelldatei, MORE stellt jeweils genau eine Bildschirmseite (24 Zeilen) einer Textdatei dar und SORT sortiert eine Datei alphabetisch.

Kommandos

In den folgenden Kommando-Beschreibungen wird auf das Ergänzen von Dateinamen mit Pfad-Namen nicht mehr getrennt eingegangen; Dateinamen werden einfach mit NAME gekennzeichnet. Wie unter CP/M kann ihnen auch eine Laufwerksbezeichnung (z.B. A:) vorausgestellt werden.

Wenn man in einer Zeile mehrere Kommandos mit Parameter-Weitergabe eingeben möchte, muß man sie mit einem senkrechten Strich voneinander trennen (Piping), z.B.:

DIR | SORT > INHALT

Diese Befehlsform legt das Disk-Directory alphabetisch sortiert im File INHALT ab. Die Anweisungsfolge

DIR | SORT | MORE

stellt dagegen jeweils eine Bildschirmseite des sortierten Inhaltsverzeichnisses auf dem Schirm dar. Dabei wird in der untersten Zeile automatisch MORE generiert, so daß mit einem Druck auf die Return-Taste die nächste Seite angezeigt werden kann.

(Bei MS-DOS-Computern stehen meist unabhängig vom nationalen Zeichensatz immer alle ASCII-Zeichen wie senkrechter Strich = hex 7C und Backslash = hex 5C zur Verfügung. Entgegen der internationalen ISO-Empfehlung werden sie nicht durch nationale Sonderzeichen ersetzt. MS-DOS verwendet grundsätzlich 8-Bit-

Codes; bei Sonderzeichen wie z.B. deutschen Umlauten ist Bit 7 = 1. Bei Datenfernübertragung mit Gegenstationen, die sich an den ISO-Standard halten, ist ggf. eine Umcodierung erforderlich!)

Folgende MS-DOS-Kommandos sind innerhalb des Betriebssystem-Kerns implementiert, der immer im Speicher verbleibt, sind also interne Kommandos:

BREAK	CHDIR/CD	CLS	COPY
CTTY	DATE	DEL	DIR
ECHO	ERASE	EXIT	FOR
GOTO	IF	MKDIR/MD	PATH
PAUSE	REM	REN(AME)	RMDIR/RD
SET	SHIFT	TIME	TYPE
VER	VERIFY	VOL(UME)	

Die folgenden Kommandos sind bei MS-DOS extern, d.h. sie funktionieren nur, wenn eine Systemdiskette im angesprochenen Laufwerk ist:

ASSIGN	BACKUP	CHKDSK	COMP
DISKCOMP	DISKCOPY	FC	FDISK
FIND	FORMAT	GRAPHICS	EXE2BIN
MODE	HEXDUMP	MORE	PRINT
PROMPT	RECOVER	RESTORE	SORT
SYS	SETLANG	TREE	

Nun aber zu den Kommandos im einzelnen. Die Verwandtschaft mit CP/M ist ganz offensichtlich nur noch recht entfernt, der Befehlssatz ist sehr umfangreich.

ASSIGN
Umleiten aller Zugriffe von einem Laufwerk auf ein anderes. ASSIGN ohne Parameter setzt alle Zuweisungen zurück. Beispiel:

```
ASSIGN D:=C:
```

BACKUP
Dient zum Anfertigen einer Sicherheitskopie des Festplatten-Inhalts auf einer entsprechenden Anzahl von Disketten. Es gibt folgende Optionen:

/S Nicht nur die Dateien des aktuellen Directory, sondern auch die aller darunter liegenden anderen kopieren
/M Nur die Dateien des aktuellen Directory kopieren, die seit dem letzten Backup verändert worden sind
/A Spezifizierte Dateien werden auf die Backup-Disk hinzukopiert
/P Kopiert so viele Dateien wie physikalisch möglich auf eine Backup-Diskette
/D: Nur die Dateien werden kopiert, die seit dem angegebenen Datum erzeugt oder verändert wurden
/T: Nur die Dateien werden kopiert, die seit der angegebenen Zeit erzeugt oder verändert wurden (vgl. TIME)

/L: Markiert im angegebenen Dateinamen, daß eine Sicherheitskopie gemacht wurde
/R Die Dateien, auf die auch schreibend zugegriffen werden kann, werden nicht mitkopiert

Mit dem Kommando IF kann nach Abschluß des Backup-Vorgangs der „Exit-Code" ausgewertet werden. Er kann vier Werte annehmen:

0 Kein Fehler aufgetreten
1 Es wurden keine Dateien gefunden
3 Das Kopieren wurde vom Benutzer abgebrochen
4 Das Kopieren wurde wegen eines Fehlers abgebrochen

Beispiele:

BACKUP C:*.COM A: Alle COM-Files werden von der Festplatte (C) nach Laufwerk A kopiert
BACKUP C:*.* A:/M Alle seit dem letzten Backup veränderten Dateien werden von C nach A kopiert

BREAK
Schaltet die Möglichkeit ein und aus, ein Kommando mit CTRL-C abzubrechen, wenn ON oder OFF folgt. BREAK allein zeigt den derzeitigen Status an. Beispiel:

BREAK ON
BREAK OFF

CHDIR oder CD
Wechselt das aktuelle Directory zum angegebenen Pfad. Beispiel:

CD \BIN \USER

CHKDSK
Liefert eine Status-Anzeige für ein Laufwerk oder eine Datei bzw. eine Dateigruppe. Mit der Option /F korrigiert MS-DOS automatisch Tippfehler z.B. bei der Laufwerks-Angabe.

CLS
Löscht den Bildschirm.

COMP
Vergleich zweier Dateien. Beispiel:

COMP A:*.TXT B: vergleicht alle TXT-Dateien auf den Laufwerken A und B

COPY
Kopieren einer Datei oder Verbinden zweier Dateien. Die Option /A sorgt dafür, daß das Kopieren von Textdateien beim Auftreten von CTRL-Z beeindet wird; mit /B können dagegen auch CTRL-Z-Zeichen mitkopiert werden. Beim Verbinden von Dateien ist /A voreingestellt, sonst /B. Beispiele:

COPY B:TEXT	Die Datei TEXT wird von B auf das voreingestellte Laufwerk kopiert
COPY *.* B:	Alle Dateien werden vom aktuellen Laufwerk auf das Laufwerk B kopiert
COPY TEXT+INFO	An die Datei TEXT wird die Datei INFO angehängt; TEXT wird dabei natürlich länger

CTTY
Damit kann man einstellen, woher MS-DOS seine Kommando-Eingaben bezieht. Zulässig sind die Namen CON (voreingestellt, Konsole), AUX, PRN und NUL.

DATE
Anzeigen und ggf. Ändern des Datums im amerikanischen Format (Monat-Tag-Jahr oder Monat/Tag/Jahr). Die Jahreszahl kann zwei- oder vierstellig eingegeben werden. Beispiele:

DATE 3-16-1985	setzt das Datum auf den 16. März 1985
DATE	zeigt das Datum an

DEL oder ERASE
Löschen von Dateien. DEL und ERASE sind gleichwertig. Beispiele:

DEL *.	löscht alle Dateien ohne Extension
DEL *.*	löscht alle Dateien mit Extension
DEL B:NAME.TMP	löscht die Datei NAME.TMP auf Laufwerk B
DEL N?M	löscht alle Dateien, deren Name aus drei Zeichen besteht, wovon das erste ein N und das letzte ein M ist

DIR
Zeigt die Dateien des aktuellen oder eines spezifizierten Directory an. Beispiele:

DIR/W	Alle Dateien des aktuellen Directory im Kompaktformat
DIR B:*.COM	Alle COM-Dateien auf Laufwerk B

DISKCOMP
Vergleicht den Inhalt zweier Disketten miteinander (funktioniert nicht bei Festplatten!). Die Option /1 vergleicht nur die erste Disk-Seite; mit /8 reduziert man den Vergleich auf acht Sektoren pro Spur, auch wenn eine 9-Sektor-Disk verwendet wird. Beispiel:

DISKCOMP A:B:

DISKCOPY
Dient zum Kopieren zweier Disketten, Spur für Spur und Sektor für Sektor; dazu müssen Quell- und Ziellaufwerk angegeben werden. (Tip: Je nach Belegung der Disk funktioniert COPY u. U. schneller!) Beispiel:

DISKCOPY A:B:

ECHO ON/OFF
Normalerweise werden die Kommandos einer Datei, die im Batch-Betrieb abläuft, parallel auf dem Bildschirm angezeigt. ECHO OFF schaltet diese Anzeige aus, ECHO ON wieder ein.

ERASE
Identisch mit dem Kommando DEL (siehe dort).

EXE2BIN
Wandelt Maschinenprogramm-Dateien vom relokatiblen EXE-Typ, wie er z.B. vom Linker des Microsoft-Assemblers Macro-86 erzeugt wird, in COM-Typ-Dateien um. Eine Datei darf nicht länger als 64 KByte sein. Das Kommando EXE2BIN NAME holt sich die Datei NAME.EXE und erzeugt daraus eine Datei NAME.COM.

EXIT
Wie in CP/M-Plus ist bei MS-DOS der Kommandoprozessor nicht resident im Arbeitsspeicher, sondern wird jeweils als Datei COMMAND.COM von der Systemdiskette geladen. Dies muß man von einem Anwenderprogramm aus nach Erscheinen des Laufwerk-Prompts (z.B. A>) sogar explizit tun, um MS-DOS-Kommandos wie z.B. DIR aufrufen zu können. Mit EXIT gelangt man anschließend zu dem vorher verlassenen Anwenderprogramm zurück.

FC
Vergleich zweier Dateien; Unterschiede werden ausgegeben. Beispiel:
FC TEST1 TEST2

FDISK
Anzeigen oder Ändern der Dateiverwaltung auf einer angeschlossenen Festplatte. Das FDISK-Programm ist menügesteuert und erlaubt es z.B., nur einen Teil der Harddisk für MS-DOS zu verwenden, einen anderen Teil jedoch für ein anderes Betriebssystem.

FIND
Zeigt alle Zeilen einer Datei an, die eine spezifizierte Zeichenfolge enthalten. Optionen: Mit /V werden umgekehrt alle Zeilen angezeigt, die den String nicht enthalten. Mit /C wird nur die Zahl der resultierenden Zeilen angezeigt. Mit /N wird jeder angezeigten Zeile ihre Nummer (in der Quelldatei) vorangestellt. Beispiele:

FIND "FOLGE" NAME1 NAME2	zeigt alle Zeilen der Dateien NAME1 und NAME2 an, die die Zeichenfolge FOLGE enthalten
DIR B:| FIND /V "ABC"	zeigt alle Dateinamen von Laufwerk B an, die die Zeichenfolge ABC enthalten

FOR...IN...DO
Schleifenanweisung zur automatischen Wiederholung von MS-DOS-Kommandos. Bei manueller Eingabe lautet die Syntax:

FOR %Zeichen IN (Kommandoparameter) DO Kommando %Zeichen

Die einzelnen Prozent-Zeichen sind für Batch-Betrieb durch doppelte (%%) zu ersetzen. „Zeichen" ist ein nichtnumerisches ASCII-Zeichen und dient als (beliebig

wählbarer) Variablenname. Das folgende Beispiel listet alle TXT-Dateien einer Diskette:

```
FOR %%i IN (*.TXT) DO TYPE %%i
```

FORMAT

Formatieren oder Löschen von Disketten. Folgende Optionen gibt es: /1 formatiert nur einseitig; /8 formatiert nur 8 statt 9 Sektoren, wobei man zusätzlich mit /O ein zu MS-DOS 1.0 kompatibles Format erzwingen kann; /S kopiert die Systemspuren auf die neue Diskette; nach /V fragt das FORMAT-Programm nach einer „Volume"-Zeichenfolge, um einer Diskette einen Namen geben zu können. Beispiel zum Formatieren einer Disk in Drive (Laufwerk) C und zum Schreiben der Systemspuren mit dem MS-DOS-Kern:

```
FORMAT C:/S
```

GOTO

Sprung zu einer bestimmten Stelle in einer Batch-Datei; als Zielangabe dient eine Zeichenfolge (Label). Beispiel:

```
:ABC        Label-Definition in der Batch-Datei
...         Weitere Anweisungen
GOTO ABC    Sprung zum Label ABC
```

GRAPHICS

Gibt die Taste PRTSC frei, um auf dem Bildschirm angezeigte Grafiken auf einem angeschlossenen (grafikfähigen) Drucker auszudrucken.

HEXDUMP

Zeigt den Inhalt einer Datei als Hex-Dump an (hexadezimaler Speicherauszug), indem in jeder Zeile zunächst die vierstellige Adresse, dann 16 Hex-Bytes und schließlich deren ASCII-Darstellung angezeigt werden. Die Datei darf maximal 64 KByte lang sein. Beispiel:

```
HEXDUMP A:NAME
```

IF

Bedingungs-Anweisung innerhalb einer Batch-Datei. Bedingung: EXIST NAME stellt fest, ob eine Datei NAME existiert. %n==String prüft, ob der im Batch-File substituierte Parameter gleich der Zeichenfolge „String" ist. Mit ERRORLEVEL läßt sich feststellen, ob ein vorhergehendes Kommando (z.B. BACKUP) einen bestimmten Exit-Code liefert. Beispiele:

```
IF NOT EXIST NAME ECHO GIBT ES NICHT     Der Text „Gibt es nicht"
                                         wird ausgegeben, wenn die Datei NAME
                                         nicht im aktuellen Directory existiert
IF ERRORLEVEL 1 GOTO ABC                 Sprung zum Label ABC, wenn der Exit-
                                         Code des vorherigen Kommandos 1 war
```

MKDIR oder MD
Hinzufügen eines neuen Directory-Pfades. Beispiel:
MD USER

MODE
Setzen von Parametern für Konsole (z.B. Bildschirmdarstellung) oder Schnittstellen. Beispiele:

MODE COM1:30,N,8,1	Stellt die V.24-Schnittstelle 1 auf 300 Baud, kein Paritätsbit (N; E=gerade, O=ungerade), 8 Datenbits, 1 Stopbit ein
MODE COM1:11	Stellt die V.24-Schnittstelle auf 110 Baud um und läßt die übrigen Parameter unverändert
MODE 40 (oder 80)	Schaltet bei angeschlossenem Farbbildschirm auf 40 (oder 80) Zeichen pro Zeile
MODE BW 80	Schaltet den Video-Ausgang auf den Farbbildschirm, diesen jedoch auf Schwarzweiß-Darstellung mit 80 Zeichen pro Zeile
MODE CO 40	Schaltet den Video-Ausgang auf den Farbbildschirm und diesen auf Farbdarstellung mit 40 Zeichen pro Zeile
MODE PTYPE=PR2300	Stellt die Steuerzeichen für den Druckertyp PR2300 ein, der in der MS-DOS-Datei PRINTCAP enthalten sein muß
MODE LPT1:80,6	Stellt den Drucker 1 auf 80 Zeichen pro Zeile und 6 Punkte Zeilenabstand ein (meist sind nur die Werte 6 oder 8 zulässig)
MODE LPT1:=COM1	Leitet die Ausgaben für Drucker 1 auf die V.24-Schnittstelle 1 um

MORE
Stellt bei Ausgaben jeweils nur einen Bildschirminhalt dar und schreibt an die unterste Bildschirmzeile wieder MORE, so daß man durch Drücken von Return die nächste Seite angezeigt erhält. Beispiel:
TYPE NAME | MORE

PATH
Ermöglicht den Zugriff auf externe Kommandos, während man sich in einem anderen Teil der hierarchischen Directory-Struktur befindet, ohne daß man den Pfad jedesmal den Kommandos voranstellen muß. Ohne Parameter wird der aktuelle Suchpfad angezeigt. Das Kommando „PATH;“ setzt den Pfad auf das aktuelle Directory zurück. Beispiel:
PATH \BIN \USER

PAUSE
Ermöglicht das Anhalten der Ausführung einer Batch-Datei mit optionaler Anzeige eines Textes. Die Ausführung kann dann mit einem Druck auf CTRL-C abgebrochen oder mit einer beliebigen anderen Taste fortgesetzt werden. Beispiel:

```
PAUSE ABBRUCH MIT CTRL-C MÖGLICH
```

PRINT
Ermöglicht wie in MP/M die Ausgabe von Textdateien auf einen Drucker, während man mit anderen Kommandos arbeitet (Spooler). Allerdings darf man die Dateinamen nicht mit Directory-Pfaden ergänzen. Folgende Optionen sind möglich: /T löscht den Puffer, /C erlaubt es, einzelne Dateien einer Dateigruppe zu überspringen; /P wählt dagegen einzelne Dateien einer Dateigruppe zum Drucken aus. Ein Beispiel:

```
PRINT /T *.TXT          Löschen des Puffers, Auswahl aller TXT-Dateien
PRINT NAME.TXT/C        Die Datei NAME soll nicht ausgegeben werden
```

PROMPT
Erlaubt das Definieren eines anderen Prompt-Zeichens statt „>". Ohne folgenden Parameter wird das normale Prompt-Zeichen definiert. Folgende spezielle Zeichen und Zeichenfolgen sind ebenfalls definierbar:

$	Dollar-Symbol, gefolgt von:	t	Aktuelle Zeit
d	Aktuelles Datum	p	Aktuelles Directory
v	MS-DOS-Versionsnr.	n	Voreingestelltes Laufwerk
g	Größer-Symbol: >	l	Kleiner-Symbol: <
b	Senkr. Strich: |	_	CR-LF (neue Zeile)
s	Leerraum	h	Backspace
e	Escape-Zeichen	f	Gleichheitszeichen: =

RECOVER
Markiert fehlerhafte Sektoren einer Diskette (diese werden dadurch später nicht mehr verwendet). Gibt man nur eine Laufwerksbezeichnung als Parameter an, so wird die gesamte Disk geprüft. Wenn ein Dateiname folgt (Wildcards „*" und „?" sind nicht zugelassen), versucht MS-DOS diese Datei soweit zu retten, wie möglich und markiert ebenfalls fehlerhafte Stellen.

REM
Erlaubt wie in Basic das Einfügen von Kommentaren mit bis zu 123 Zeichen Länge in MS-DOS-Kommandozeilen.

REN (RENAME)
Umbenennen einer Datei oder (mit Wildcards „*" oder „?") einer Dateigruppe, wobei die dadurch ignorierten Teile des ursprünglichen Namens unverändert bleiben. Beispiel zum Umbenennen von NAME1 in NAME2:

```
REN NAME1 NAME2
```

RESTORE
Umladen von Dateien von Backup-Disketten auf das Hard-Disk-Laufwerk. Hierbei gibt es Optionen und Exit-Codes wie beim Kommando BACKUP. Beispiel:

```
RESTORE A:C:
```

RMDIR
Dient zum Löschen von Directory-Pfaden. Sicherheitshalber wird das Löschen nur von solchen Directories zugelassen, die weder Dateien noch Subdirectories enthalten. Beispiel:

```
RMDIR \USER\HANS
```

SET
Erlaubt in Programmen und Batch-Dateien das automatische Ersetzen einer über die Tastatur eingegebenen Zeichenfolge durch eine andere. Innerhalb einer Batch-Datei ist die Zeichenfolge mit Prozentzeichen einzurahmen (z.B. %STRING%). Wenn man beispielsweise aus jeder Eingabe von ABC die Folge DEF machen will, so gibt man nur ein:

```
SET ABC=DEF
```

SETLANG
Da es in den einzelnen Ländern unterschiedliche nationale Zeichensätze und damit auch unterschiedliche Tastatur-Belegungen gibt, wird bei MS-DOS-Rechnern eine Tastatur-Umcodier-Tabelle im RAM angelegt. Sie wird mit SETLANG von der Systemdiskette geladen, wobei der Name der Datei mit der gewünschten Belegung anzugeben ist (diese sollte natürlich mit der Tastatur-Beschriftung übereinstimmen). Beispiel zur Auswahl der deutschen Belegung:

```
SETLANG KEYBGR
```

SHIFT
Da innerhalb einer Batch-Datei normalerweise maximal 10 Substitutions-Parameter verwendet werden können (%0...%9), kann man mit wiederholten SHIFT-Anweisungen ihre Numerierung verschieben. Beispielsweise existiere zunächst folgende Zuweisung:

%0 = "abc"
%1 = "def"
%2 = "ghi"
%3...9 nicht belegt
Nach dem Kommando SHIFT sind die Parameter um einen verschoben:
%0 = "def"
%1 = "ghi"
%2...9 nicht belegt

Es wurde also für einen zusätzlichen Parameter Platz geschaffen.

SORT
Sortieren einer Datei, normalerweise alphabetisch. (Vorsicht: Die Anordnung der deutschen Umlaute im Zeichensatz entspricht bei MS-DOS weder der ASCII-

Norm noch einer sinnvollen alphabetischen Reihenfolge!) Mit der Option /R wird in umgekehrter Reihenfolge sortiert, mit /+ kann man das Sortieren bei einer bestimmten Zeilennummer beginnen lassen. In folgendem Beispiel wird die Datei NAME ab ihrer zweiten Zeile sortiert, so daß der Platz der Text-Überschrift unverändert bleibt:

SORT NAME /+2

SYS
Dient zum Schreiben der MS-DOS-Systemdateien MSDOS.SYS und IO.SYS auf eine Diskette, bei der die Systemspuren noch frei sind (!). Beispiel:

SYS B:

TIME
Anzeigen und Setzen der Uhrzeit (24 Stunden) im Format hh:mm:ss. Beispiele:

TIME	zeigt die Uhrzeit an
TIME 15	setzt die Uhr auf 15 Uhr
TIME 15:30	setzt die Uhr auf 15:30 Uhr

TREE
Anzeigen aller Directories und Pfade eines Laufwerks. Mit der Option /F werden auch die Dateinamen selbst gelistet. Beispiel:

TREE A:/F

TYPE
Anzeigen des Inhalts einer Textdatei. Beispiel:

TYPE NAME.TXT

VER
Zeigt die Versionsnummer des Betriebssystems auf dem Bildschirm an.

VERIFY ON/OFF
Nach dem Kommando VERIFY ON werden alle Dateien, die auf Diskette oder Festplatte geschrieben werden, zur Sicherheit noch einmal gelesen und geprüft. (Dadurch dauert das Abspeichern natürlich etwas länger.) Dieses Prüflesen läßt sich mit VERIFY OFF wieder ausschalten.

VOL oder VOLUME
Zeigt den Titel einer Diskette an; optional kann ein Laufwerk benannt werden. Beispiel:

VOL B:

Der Bildschirm-Editor EDIT

Zu MS-DOS 2.1 gibt es zwei verschiedene Texteditor-Programme, nämlich EDIT und EDLIN. Während EDIT bildschirmorientiert arbeitet, ist EDLIN wie der CP/M-Editor ED ein zeilenorientiertes Programm.

Aufgerufen wird der Bildschirm-Editor im Format EDIT NAME, wobei NAME die zu bearbeitende oder zu erstellende Datei ist und auch einen Directory-Pfad enthalten darf. Vor den Namen bzw. Pfad können Optionen in beliebiger Reihenfolge geschrieben werden:

/B Beim Aufruf des Editors wird automatisch eine Sicherheitskopie der alten Textversion mit der Extension .BAK angefertigt

/R Damit kann man eine Datei vor versehentlichem Ändern schützen, wenn man sie ohnehin nur ansehen möchte

/T Überall dort, wo mehrere Leerräume eingegeben werden, werden sie automatisch durch Tabulator-Zeichen ersetzt, um Speicherplatz zu sparen

Nach dem Aufruf von EDIT stehen 21 Zeilen mit je 80 Zeichen zur Verfügung, in denen sich Text eingeben und der Cursor frei bewegen läßt; das geschieht mit den Pfeiltasten. Die oberste Bildschirmzeile zeigt den Dateinamen des zu bearbeitenden Textes an, die zweite ist für die Eingabe von Kommandos reserviert, in der dritten werden die Tabulator-Positionen angezeigt. Das eigentliche Textfenster von 21 Zeilen läßt sich nach oben und unten verschieben, so daß man den gesamten Text bearbeiten kann. Insgesamt gibt es (außer den Pfeiltasten) folgende Tastenfunktionen:

HOME	Cursor an oberste Textzeile setzen
END	Cursor an letzte (unterste) Zeile setzen
PGUP	Einen Bildschirm nach oben
PGDN	Einen Bildschirm nach unten
F1	Kommando-Eingabe (z.B. Suchen)
Shift-F1	Editor verlassen (Abort)
F2	Gelöschte Zeile wiederherstellen
Shift-F2	Zeile löschen
F3	Zeilen verketten
Shift-F3	Zeilen aufspalten
F4	Zeilenende markieren
Shift-F4	Zeilenbeginn markieren
F5	Text abspeichern
Shift-F5	Text abspeichern und Editor verlassen
F6	Nächste Zeile
Shift-F6	Zeile ab dem Cursor bis zum Ende löschen
F7	Zu einer Markierung springen
Shift-F7	Marke an Cursorposition eingeben
F9	Eine Zeile nach unten
Shift-F9	Eine Zeile nach oben
F10	Halben Bildschirm nach unten
Shift-F10	Halben Bildschirm nach oben
BS, CTRL-H	Backspace, ein Zeichen rückwärts
TAB, CTRL-I	Tabulator nach rechts
Shift-TAB	Tabulator nach links

CTRL-K	Zeile bis zum Ende löschen
CTRL-R	Zeile wiederherstellen
INS	Insert-Modus: Eingetippte Zeichen ab Cursorpos. einfügen
DEL	Zeichen löschen
CR	Zeile einfügen; Kommando-Modus: Kommando ausführen

Mit F1 gelangt man in den Kommando-Eingabe-Modus. Dann kann man in der zweiten Bildschirmzeile, die oberhalb des eigentlichen Textfensters ist, folgende Befehle eingeben:

F8	Vom Cursor an abwärts eine Zeichenfolge suchen (zuerst F1, dann Zeichenfolge, dann F8 eingeben!)
Shift-F8	Vom Cursor an aufwärts eine Zeichenfolge suchen
DELETE	Text zwischen Cursor und Markierung löschen
FILE NAME	Vorübergehend andere Datei NAME editieren; Rückkehr zur alten mit Shift-F1 oder Shift-F5
GOTO nn	Cursor an die Zeile mit der Nummer nn setzen

Die maximale Zeilenlänge beträgt bei EDIT 80 Zeichen. Falls dies nicht ausreicht, muß man mit EDLIN arbeiten.

Der Zeilen-Editor EDLIN

Der Zeilen-Editor wird mit EDLIN NAME aufgerufen; die Verwendung eines Directory-Pfads ist nicht möglich. Beim Start wird automatisch eine eventuell vorhandene Sicherheitskopie NAME.BAK gelöscht (!). Die Verwendung einer .BAK-Datei als Name ist nicht zulässig.

Das EDLIN-Prompt-Zeichen zur Markierung der „aktuellen" Zeile ist ein Stern, der Cursor ein Unterstreichungs-Symbol. Alle Zeilen werden mit einer vorangestellten Zahl, der ein Doppelpunkt folgt, angezeigt.

Falls die Datei NAME länger als drei Viertel des freien Speicherplatzes ist, wird zunächst nur ihr erster Teil ins RAM geladen; an den nächsten Teil gelangt man jeweils mit besonderen Kommandos zum Abspeichern und Anhängen von Zeilen.

Bei EDLIN darf eine Zeile bis zu 255 Zeichen umfassen, und die zu bearbeitende Datei darf nahezu beliebig lang sein. Kommandos dürfen wahlweise in Groß- oder Kleinschrift eingegeben werden. Bei den Kommandos zur Bearbeitung von Text muß die gewünschte Zeilennummer entweder als Zahl oder, wenn die zuletzt bearbeitete gemeint ist, als Punkt (.) vorangestellt werden. Folgende Kommandos gibt es bei EDLIN, wobei Zeilennummern mit n bezeichnet sind (ein Leerraum zwischen n und dem Kommando ist nicht nötig):

n
Durch Eingabe einer Zahl wird die Zeile mit dieser Nummer die aktuelle Zeile.

n A
An den Text im Speicher werden n Zeilen von der Disk-Datei angehängt. Fehlt n, so werden solange Zeilen angehängt, bis der Speicher zu 75 % voll ist (Append).

n1,n2,n,m C
Die Zeilen mit den Nummern n1 bis n2 werden ab Zeile n genau m-mal kopiert; wenn m fehlt, nur einmal. Wenn n1 oder n2 fehlen, wird statt dessen jeweils die aktuelle Zeile angenommen (Copy).

n1,n2 D
Die Zeilen n1 bis n2 werden gelöscht. Fehlen n1 oder n2, wird jeweils die aktuelle Zeile angenommen (Delete).

E
Beendet das EDLIN-Programm und speichert die aktualisierte Version des Textes auf Diskette.

nI oder .I oder # I
Eingabe von Text. nI fügt den neuen Text vor der Zeilennummer n ein, .I vor der aktuellen Zeile, # I am Ende des bisherigen Textes. Die Eingabe wird mit CTRL-Z beendet oder mit CTRL-C abgebrochen (Insert).

n1,n2 L
Die Zeilen n1 bis n2 werden auf dem Bildschirm gelistet. Fehlt n1 und/oder n2, werden jeweils 23 Zeilen gelistet.

n1,n2,m M
Verschieben von Zeilen im Text von n1 bis n2 nach m, wobei die Zeilen natürlich umnumeriert werden. Anschließend gilt die erste verschobene Zeile als aktuelle Zeile. Fehlt eine der Angaben n1, n2 oder m, so wird dafür die vorherige aktuelle Zeile angenommen (Move).

n1,n2 P
Funktioniert wie L, allerdings wird die aktuelle Zeile gleich der letzten angezeigten gesetzt (Page).

Q
EDLIN wird verlassen, ohne (!) den erstellten Text auf Diskette zu retten, d.h. er ist dann verloren (Quit).

n1,n2 R string1 string2
Ersetzen der Zeichenfolge „string1“, die mit CTRL-Z abgeschlossen sein muß, durch die Zeichenfolge string2. Fehlt string2, braucht string1 nicht mit CTRL-Z beendet werden; dann wird string1 überall dort, wo er zwischen n1 und n2 auftritt, gelöscht. Stellt man vor R ein Fragezeichen (?R), so fragt EDLIN bei jedem Auftreten von string1 „OK?“, so daß man die Wahl hat, ob er ersetzt bzw. gelöscht werden soll oder nicht. Fehlen n1 und n2, wird der Text ab der aktuellen Zeile durchsucht (Replace).

n1,n2 S string
Durchsucht den Zeilenbereich n1 bis n2 nach der Zeichenfolge „string". Beim ersten Auftreten wird die Suche beendet und die Zeile (die dann die aktuelle Zeile ist) angezeigt. Stellt man vor S ein Fragezeichen (?S), so wird dagegen die Zeile und „OK?" angezeigt, so daß man die Möglichkeit hat, weiterzusuchen. Fehlen n1 und n2, so wird der Text ab der aktuellen Zeile durchsucht (Search).

n T name
Einfügen der Disk-Datei „name" vor die Zeile n. Fehlt n, wird die aktuelle Zeile angenommen. Die einzufügende Datei muß im gleichen Directory zu finden sein wie die gerade bearbeitete.

n W
Schreibt ab Zeile 1 genau n Zeilen auf Diskette und löscht sie dann im Arbeitsspeicher; wenn n fehlt, wird solange geschrieben, bis mindestens 25% freies RAM zur Verfügung steht (Write; vgl. Kommando A).

Solange man das Editieren einer Zeile noch nicht beendet hat, hält EDLIN im Speicher eine Kopie der ursprünglichen aktuellen Zeile bereit. Zusammen mit den besonderen Tasten der MS-DOS-Tastatur stehen dadurch eine Reihe von nützlichen Kommandos zur Verfügung, die man innerhalb einer zu editierenden Zeile verwenden kann:

F1	Kopiert ein Zeichen von der ursprünglichen in die neue Zeile
F2c	Kopiert Zeichen bis zum Zeichen c von der ursprünglichen in die neue Zeile
F3	Kopiert die ursprüngliche in die neue Zeile
F4c	Löscht alle Zeichen in der ursprünglichen Zeile bis zum ersten Auftreten des Zeichens c
F5	Die ursprüngliche Zeile wird auf den neuen Stand gebracht
DEL	Löscht ein Zeichen in der bisherigen Zeile
ESC	Die editierte Zeile wird gelöscht, die ursprüngliche bleibt unverändert
INS	Ein- und Ausschalten des Insert-Modus (Einfügen/Überschreiben)

Der Debugger DEBUG

Mit MS-DOS wird ein Monitorprogramm namens DEBUG geliefert, das zum Prüfen von Maschinenprogrammen sowie zum Eingeben kurzer Routinen geeignet ist. Es beinhaltet u.a. auch einen Line-by-Line-Assembler sowie einen Disassembler für den 8086-Befehlssatz. Der Aufruf erfolgt mit DEBUG gefolgt von einem Dateinamen sowie optional einer Argumentliste, z.B. eines Dateinamens, den das zu testende Programm als Argument benötigt. Die Kommando-Eingabe kann wahlweise in Groß- oder Kleinbuchstaben erfolgen. Eine unschöne Einschränkung: EXE-Dateien können zwar geladen, nicht aber mit W wieder abgespeichert werden (!).

Die Eingabe von Adressen (in der folgenden Kommandotabelle mit a gekennzeichnet) ist hexadezimal in folgenden Formaten erlaubt:

DS:0400	Segmentregister-Name und Offset-Adresse
8C0:0225	Speichersegment und Offset-Adresse
225	Offset-Adresse im voreingestellten Segment

Adressenbereiche (r) kann man folgendermaßen spezifizieren:

a L z	z Bytes ab Adresse a (z = Länge)
a1,a2	Bereich von Adresse a1 bis a2 (auch Leerraum statt Komma zulässig)

Bytes (b) kann man ein- oder zweistellig hexadezimal eingeben. Nun jedoch zu den Kommandos des DEBUG-Programms; optionale Angaben stehen jeweils in eckigen Klammern.

A <a>
Ab der Adresse a oder, falls diese nicht angegeben wurde, ab der aktuellen Adresse mnemonische Befehle in den Speicher eingeben. Labels sind nicht zulässig. Zur Unterscheidung, ob das Argument eines Befehls ein Byte oder ein 16-Bit-Wort ist, muß man es ggf. mit „BYTE PTR" (abgekürzt BY) bzw. mit „WORD PTR" (WO) spezifizieren. Für „Far Return" muß man RETF, für Segment-Anwahl CS:, DS:, ES: und SS: als Mnemonics eingeben. Die Bezeichnungen NEAR (kurz NE) und FAR (kurz FA) vor Sprungadressen erzwingen die Assemblierung von „nahen" und „weiten" Sprüngen, andernfalls wird, wo immer möglich, ein NEAR angenommen. Ein Argument in eckigen Klammern bedeutet, daß es sich um eine Adresse handelt, andernfalls um einen direkten Operand (Immediate). Zum Ablegen von Bytes und Worten gibt es die Pseudo-Befehle DB und DW. Nach jeder eingegebenen Zeile wird sofort der erzeugte Objektcode angezeigt.

C r,a
Vergleich zweier Speicherbereiche. Der Bereich r wird mit dem bei Adresse a beginnenden gleich langen verglichen (Compare).

D r oder D <a>
Speicherbereich r hexadezimal und als ASCII-Übersetzung anzeigen. Falls statt r nur eine Adresse a genannt wird, werden 128 Bytes dargestellt. Fehlt auch a, zeigt DEBUG 128 Bytes ab der aktuellen Adresse an (Display).

E a <,b> <,b...>
Ab der Adresse a werden ein oder mehrere Bytes b überschrieben (Enter). Die Bytes sind voneinander entweder mit Kommas oder mit Leerräumen zu trennen.

F r,b <,b...>
Füllt den Speicherbereich r mit dem Byte b oder einer Folge von mehreren Bytes (Fill).

G <=a> <,a...>
Start eines Programms an der Adresse a; fehlt diese Angabe, erfolgt der Start an der mit den CS- und IP-Registern spezifizierten Adresse. Weitere Adressen hinter der Startadresse (max. 10) dienen als Breakpoints, d.h. das Programm hält dort jeweils an (Go).

H a,b
Errechnet die Summe und Differenz zweier hexadezimaler Werte a und b, also a+b und a−b.

I a
Zeigt den hexadezimalen Wert vom I/O-Port an der Adresse a an (Input).

L <a>
Laden eines Programms entweder ab der Adresse a oder, falls diese nicht angegeben ist, ab Adresse 0100 im aktuellen, durch das Register CS spezifizierten Segment (Load). Folgender Befehl lädt neun Sektoren ab Sektor 5 (nämlich das Directory) ab Adresse 0100:

L CS:0100 0 5 9

Dabei steht 0 für Laufwerk A.

M r,a
Kopiert den Speicherbereich r an die Adresse a. Dabei ist es auch zulässig, daß sich Quell- und Zielbereich überlappen oder in unterschiedlichen Segmenten stehen (Move).

N name
Definieren eines Namens zum Laden oder Abspeichern von Dateien; er steht dann im File-Control-Block ab Adresse CS:5C.

O a,b
Ausgabe des Werts b an den I/O-Port mit der Adresse a (Output).

Q
Verlassen des Debuggers, ohne das eventuell geänderte Programm im Speicher auf Diskette zu retten (Quit).

R <rs>
Anzeigen und ev. Ändern des Registers mit dem Namen rs oder, falls nicht angegeben, aller Register der CPU sowie des disassemblierten nächsten Befehls.

RF
Anzeigen und ev. Ändern der Status-Flags der CPU.

S r,b <,b...>
Suche nach einem Byte b oder einer Bytefolge im Adressenbereich r, wobei jeweils die Anfangsadresse des Auftretens angezeigt wird (Search).

T <=a> <,b>
Führt b Befehle (oder, falls b fehlt, einen Befehl) ab der Adresse a aus, bzw. bei Fehlen von a ab der durch die CS- und IP-Register spezifizierten Adresse. Anschließend werden die CPU-Register und der nächste disassemblierte Befehl des Programms angezeigt (Trace).

U r oder U <a>
Disassemblieren des Speicherbereichs r oder 32 Bytes ab Adresse a (bzw. bei fehlender Angabe ab CS:IP; Unassemble).

W <a>
Schreiben des Speicherbereichs mit der Anfangsadresse (oder, falls a fehlt, ab CS:0100) und der Byteanzahl in den Registern BX:CX. Der Bereich wird beim Laden einer Datei automatisch richtig gesetzt (Write). Wie bei L kann man auch einzelne Disk-Sektoren schreiben, z.B.:

W CS:100 0 5 9 (schreibt die Directory-Spur zurück)

Ein weiteres Hilfsmittel ist in MS-DOS neben DEBUG der Linker (LINK). Er arbeitet menügesteuert und braucht deshalb hier nicht näher erläutert zu werden. Sein Zweck ist es vor allem, mehrere vom Microsoft-Assembler Macro-86 erzeugten relokatiblen Objektcode-Files zu einem einzigen Gesamtprogramm zusammenzubinden. Resultat ist wiederum ein relokatibles Objektprogramm, ferner ein Kontroll-Listing mit eventuellen Fehlermeldungen und Verweise auf externe Labels.

MS-DOS-Fehlermeldungen

Unter MS-DOS werden Fehlermeldungen wie bei CP/M in deutschem oder englischem Klartext ausgegeben. Fehler, die mit Ein- und Ausgabe-Operationen zu tun haben, erscheinen stets im folgenden Format:

(Typ) ERROR READING/WRITING (Gerät)
ABORT, RETRY, IGNORE?

Nach einer solchen Meldung kann man die Operation mit A abbrechen, mit R erneut versuchen und mit I den Fehler einfach ignorieren und somit u.U. falsche Daten in Kauf nehmen. Der Fehlertyp kann sein:

bad call format	Fehler beim Aufruf
bad command	Kommando falsch geschrieben
bad unit	Gerät nicht aktiv
data	Fehler in den Daten selbst
disk	Laufwerk nicht aktiv
no paper	Drucker meldet Papiermangel
non-dos disk	Diskette ist nicht für MS-DOS geeignet
not ready	Gerät ist nicht bereit
read fault	Fehler beim Lesen von Disk
sector not found	Disk-Sektor wurde nicht gefunden
seek	Unlogisches Adressenfeld auf der Diskette
write fault	Fehler beim Schreiben auf Disk
write protect	Diskette ist schreibgeschützt

Zeichensatz IBM-PC-kompatibler MS-DOS-Computer

Da die Syntax zahlreicher Systemprogramme Zeichen wie eckige Klammern, senkrechte Striche und andere erfordert, die nur im (amerikanischen) ASCII-Zeichensatz enthalten sind und nicht in der an den deutschen Zeichensatz angepaßten ISO-7-Bit-Norm, wurde von IBM ein 8-Bit-Code gewählt, der mit 255 maximal mögli-

Abb. 4.1.2.2 MS-DOS wurde ursprünglich für den IBM-PC entwickelt, steht inzwischen aber auch für viele andere Rechner angepaßt zur Verfügung

chen Zeichen sowohl nationale Sonderzeichen wie z. B. deutsche Umlaute als auch alle Original-ASCII-Zeichen darstellen kann. Es muß betont werden, daß diese Lösung bestenfalls dem internen IBM-Standard, aber keiner offiziellen Norm entspricht *(Abb. 4.1.2.2)*.

Grundsätzlich entsprechen dabei alle Codes mit dem höchstwertigen Bit (Bit 7) = 0 dem ASCII-Standard. Allerdings sind auch die eigentlich für Steuerzeichen vorgesehenen Codes hex 00...1F (dezimal 0...31) mit abbildbaren Blockgrafik-Zeichen belegt; sie werden nur von der I/O-Treiber-Routine gewöhnlich abgefangen und sind z. B. über die Basic-Funktion CHR$ zugänglich. Nationale Sonderzeichen finden sich im Codebereich hex 80...FF (dezimal 128...255). Wohlgemerkt: Der Zeichengenerator enthält stets die nationalen Sonderzeichen aller Länder auf einmal. Mit dem MS-DOS-Kommando SETLANG wird also nicht etwa der Zeichengenerator, sondern lediglich die Tastatur-Tabelle mit den landesspezifischen Zuordnungen geladen (!), damit sie mit der tatsächlichen Tastenkappen-Beschriftung übereinstimmt.

Da eine Datenkommunikation z.B. über Telefon-Modems normalerweise immer mit dem 7-Bit-ISO-Code erfolgt, ist bei IBM-PC-kompatiblen Rechnern erst eine Umcodierung nationaler Sonderzeichen auf den im ISO-Standard vorgesehenen Code erforderlich. Bei den deutschen Umlauten ergeben sich folgende hexadezimale Codes im IBM-Zeichensatz und in der nationalen ISO-Norm:

Zeichen	IBM	ISO	Zeichen	IBM	ISO
§	15	40	ß	E1	7E
Ä	8E	5B	ä	84	7B
Ö	99	5C	ö	94	7C
Ü	9A	5D	ü	81	7D

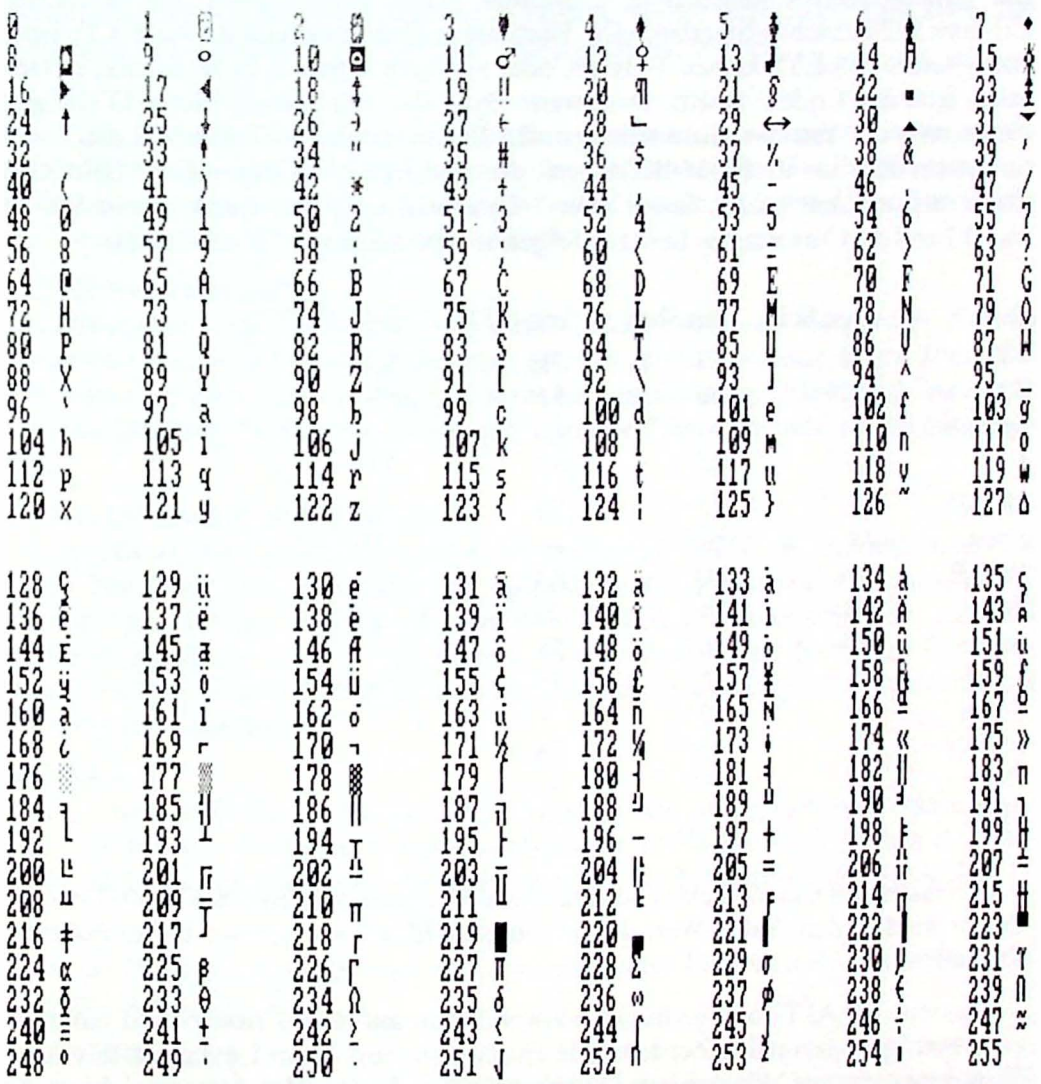


Abb. 4.1.2.3 Zeichensatz bei IBM-PC-kompatiblen Computern

Abb. 4.1.2.3 enthält den vollständigen Zeichensatz des IBM-PC. Schließlich sei noch erwähnt, daß die meisten Sortierprogramme die korrekte Einordnung nationaler Sonderzeichen nicht beherrschen. Obwohl ein Ä eigentlich unter A eingeordnet werden müßte, wird es von solchen Sortierroutinen gewöhnlich nach dem Z eingefügt. Dieses Problem tritt im IBM- wie im ISO-Code gleichermaßen auf.

Tastatur-Scan-Codes

Die Tastatur IBM-PC-kompatibler Computer besitzt mehr Tasten, als es im ASCII- bzw. DIN-Zeichen-Standard gibt. Bestimmte Tasten liefern deshalb z.B. beim Basic-Befehl INKEY$ keinen 1-Byte-Code, sondern einen 2-Byte-String, dessen erstes Byte den Code 0 besitzt. Das zweite Byte, der sogenannte Scan-Code, gibt die Position der Taste an. (Intern liefern alle Tasten solche Scan-Codes; die Tastatur-Treiber-Routine im ROM-BIOS bzw. der nationale Treiber KEYBGR.COM macht aus den „normalen" Tasten aber 1-Bytes-Codes.) Die Funktionstasten F1 bis F10 und die Cursortasten besitzen folgende hexadezimale Scan-Codes:

Taste	ohne Shift	mit Shift	mit CTRL	mit ALT
F1	3B	54	5E	68
F2	3C	55	5F	69
usw. bis ...				
F10	43	5C	67	71
HOME	47	"7"	77	
oben	48	"8"	—	
PGUP	49	"9"	84	
links	4B	"4"	73	
rechts	4D	"6"	74	
END	4F	"1"	75	
unten	50	"2"	—	
PGDN	51	"3"	76	
INS	52	"0"	—	
DEL	53	"."	—	

Die Tasten des Cursor-Feldes liefern, wenn sie zusammen mit der Shift-Taste gedrückt werden, den ASCII-Wert der jeweiligen Ziffer bzw. bei der DEL-Taste des Punktes.

Hält man die ALT-Taste gedrückt, während man auf dem Cursor-Feld einen bis zu dreistelligen dezimalen Zeichencode eingibt, so wird beim Loslassen der ALT-Taste genau dieser Code erzeugt. Dadurch sind auch jene Zeichen erreichbar, für die es keine eigenen Tasten gibt.

Die Datei CONFIG.SYS

Beim Booten (d.h. bei Laden des Betriebssystems nach dem Einschalten des Rechners) sieht MS-DOS automatisch nach, ob eine Datei CONFIG.SYS mit Konfigurations-Daten vorhanden ist. Darin können folgende Kommandos enthalten sein:

BREAK=ON
Wenn BREAK=ON gesetzt wird, sieht MS-DOS bei jedem Systemaufruf erst nach, ob der Benutzer auf der Tastatur CTRL-C gedrückt hat. BREAK=OFF ist der Normalzustand.

BUFFERS=3
Nach dem Booten reserviert MS-DOS normalerweise zwei Disk-Puffer. Diese Anzahl kann in CONFIG.SYS mit BUFFERS geändert werden.

COUNTRY=049
Setzt das Datumsformat und das Währungssymbol (das von manchen Anwenderprogrammen abgefragt wird). Dabei ist die Zahl hinter dem Gleichheitszeichen gewöhnlich die internationale Telefon-Vorwahl-Nummer des jeweiligen Landes, für die Bundesrepublik Deutschland also 049.

DEVICE=ANSI.SYS
Installiert den (externen) ANSI-Terminal-Treiber mit Escape-Sequenzen, die vom American National Standards Institute (ANSI) genormt wurden. Damit ist es möglich, ohne großen Aufwand den Cursor zu positionieren, Display-Attribute (z.B. invers, blinkend, Farbe) zu setzen und einzelne Tasten umzudefinieren (siehe unten).

DEVICE=VDISK.SYS 120 256 8
Dieses nur ab der MS-DOS-Version 3.1 mögliche CONFIG-Kommando reserviert einen Teil des Arbeitsspeichers als sogenannte RAM-Diskette. Als Laufwerksbezeichnung wählt das (externe) Kommando VDISK.SYS automatisch den nächsten freien Buchstaben, z.B. C: bei einem 2-Floppy-System ohne Festplatte. Die Parameter bedeuten hier: 120 KByte Größe, logische Sektorlänge 256 Byte, maximal 8 Directory-Einträge.

FCBS=4,2
Nur ab MS-DOS 3.1 verfügbar. Setzt die maximale Zahl gleichzeitig offener File-Control-Blöcke (FCBs) auf vier und die jener FCBs, die beim Öffnen zu vieler FCBs nicht automatisch geschlossen werden, auf 2 (Normalwert: FCB = 4,2).

FILES=6
Setzt die Zahl der gleichzeitig möglichen geöffneten Dateien auf 6 (Normalwert ist 8; 6 ist das Minimum).

LASTDRIVE=B
Verhindert den Zugriff auf Disk-Laufwerke oberhalb von B:, auch wenn solche installiert sind.

PATH=\BIN
Falls MS-DOS externe Befehle nicht auf dem Standard-Laufwerk findet, sucht das Betriebssystem automatisch auch immer im Sub-Directory BIN nach ihnen. (Der PATH-Befehl darf auch jederzeit vom Benutzer eingegeben bzw. geändert werden.)

Der Terminal-Treiber ANSI.SYS

Wenn in der Datei CONFIG.SYS der Aufruf DEVICE=ANSI.SYS auftritt, interpretiert MS-DOS bestimmte Escape-Zeichensequenzen für Steuerzwecke — allerdings nur dann, wenn sie vom Anwenderprogramm über den MS-DOS-Funktions-Interrupt INT 21H ausgegeben werden (also nicht bei Ausgaben über das ROM-BIOS). In der folgenden Aufstellung stehen y und x für dezimale, in ASCII auszugebende Zahlen. Allen folgenden Codesequenzen sind jeweils die beiden Zeichen ESC [voranzustellen (Codes 1B 5A):

ESC [	Wirkung
y;xH	Cursor an Zeile y und Spalte x stellen
yA	Cursor um y Zeilen aufwärts
yB	Cursor um y Zeilen abwärts
xC	Cursor um x Zeichen nach rechts
xD	Cursor um x Zeichen nach links
y;xf	Cursor an Zeile y und Spalte x stellen
6n	Liefert ESC [y;xR mit Cursorposition = x,y
s	Speichert die momentane Cursorposition
u	Stellt die Cursorposition nach ESC [s wieder her
2J	Löscht den gesamten Bildschirm
K	Löscht den Bildschirm ab der Cursorposition
x;...ym	Setzt ein oder mehrere Display-Attribute x...y für die folgenden Zeichen: 0 = normal, 1 = hervorgehoben, 4 = unterstrichen (nur mit Monochrom-Karte!), 5 = blinkend, 7 = invers, 8 = unsichtbar; 30...37 = Vordergrundfarbe, 40...47 = Hintergrundfarbe
=xh	Setzt Darstellungsmodus: 0 = 40 × 25 schwarzweiß, 1 = 40 × 25 farbig, 2 = 80 × 25 schwarzweiß, 3 = 80 × 25 farbig, 4 = 320 × 200 farbig, 5 = 320 × 200 schwarzweiß, 6 = 640 × 200 schwarzweiß, 7 = Auto-CR am Zeilenende einschalten
=xl	Rücksetzen des Modus x wie bei ESC [=xh
x;yp	Ersetzt Tastencode x durch y, z. B. A zu Q mit ESC [65;81p

Die letztgenannte Funktion läßt auch das Belegen von Tasten mit ganzen Strings zu. Der String „dir“ plus ein Return-Zeichen wird folgendermaßen auf Taste F10 gelegt:

ESC [0;68;“dir“;13p

Hierbei ist 0;68 der erweiterte Code der Funktionstaste F10 (der sogenannte Scan-Code).

MS-DOS-3.1

1985 brachte Microsoft die erweiterte Version 3.1 des Betriebssystems MS-DOS heraus (die bisherigen Abhandlungen bezogen sich auf die Version 2.11; MS-DOS 3.1 ist hierzu jedoch vollständig kompatibel). An dieser Stelle muß noch bemerkt werden, daß von den einzelnen Computer-Herstellern recht unterschiedliche Versionen von MS-DOS 2.11 ausgeliefert wurden und werden. Beispielsweise fehlt oft der Video-File-Editor EDIT. Ähnliche Unterschiede gibt es auch bei den Implementationen von MS-DOS 3.1, deshalb ist die folgende Aufstellung neuer Features von Version 3.1, wie sie z. B. von Zenith ausgeliefert wird, nur als Anhaltspunkt aufzufassen. Alle hier aufgeführten neuen Kommandos sind extern, also Programme auf der Systemdiskette.

ATTRIB
Erlaubt es, einzelne Dateien einer Diskette mit einem Schreibschutz zu versehen (Read-Only-Attribut). ATTRIB NAME +R setzt den Schreibschutz, ATTRIB NAME −R hebt ihn wieder auf, ATTRIB NAME ohne Parameter zeigt den momentanen Status an.

GRAFTABL
Lädt den Zeichensatz mit den Codes 128 bis 255 in den Arbeitsspeicher, damit in den Grafik-Modi der Farbgrafik-Karte auch mit Umlauten und Sonderzeichen gearbeitet werden kann, und setzt den Vektor INT 1FH auf diesen Zeichensatz.

JOIN
Ersetzt vorübergehend eine Laufwerksbezeichnung durch einen Pfadnamen. Der Switch /D hebt die Zuordnung wieder auf. So kann man mit JOIN B: A: BIN dafür sorgen, daß anschließend unter der Bezeichnung B: der Baum BIN von Laufwerk A angesprochen wird. JOIN B: /D stellt den vorherigen Zustand wieder her.

LABEL
Dient zum Vergeben eines Volume-Namens mit bis zu elf Zeichen an eine Diskette (er erscheint nach DIR im Inhaltsverzeichnis). LABEL B:Müller gibt der Diskette in Laufwerk B den Namen Müller. Gibt man LABEL ohne Parameter ein, fragt das Programm nach dem zu vergebenden Volume-Namen.

SHARE
Erlaubt die Verwendung eines lokalen Netzwerks und reserviert Speicherplatz als Puffer für Netzwerk-Zugriffe. SHARE /F:4096 reserviert 4096 Bytes als Puffer (Normalwert ist 2048). SHARE /L:10 limitiert die Zahl der Zugriffe auf eine Datei pro Netzwerk-Zugriff (Normalwert ist 20).

SUBST
Gegenstück zu JOIN: Erlaubt es mit einem Laufwerksnamen, einen Pfad anzusprechen, da manche Anwenderprogramme die Eingabe von Pfadnamen nicht zulassen. SUBST D: A: BIN sorgt dafür, daß auf den Pfad A: BIN zugegriffen wird, wenn das (nicht wirklich vorhandene) Laufwerk D benannt wird. SUBST D: /D stellt den vorherigen Zustand wieder her.

Darüber hinaus wurden einige bisherige Anweisungen verbessert: LINK unterstützt jetzt Overlays (das bedeutet, daß ein Programm länger als der zur Verfügung stehende Speicherplatz sein darf, indem Teile nachgeladen werden können) und besitzt sechs neue Parameter-„Switches". FORMAT erlaubt es mit dem „Switch" /4, Disketten im 360-KByte-Format (also IBM-PC-kompatibel) in einem IBM-AT-kompatiblen 1,2-MByte-Laufwerk zu formatieren, und mit dem Switch /B, Platz für Systemdateien früherer MS-DOS-Versionen auf der Diskette zu reservieren. Und das Spooler-Programm PRINT besitzt sechs neue Switches für Puffergröße, Zeitscheiben-Einteilung und andere Parameter.

MS-DOS für Programmierer

Das Betriebssystem MS-DOS sorgt auch dafür, daß dem Programmierer alle wichtigen Funktionen zur Ein- und Ausgabe von Zeichen sowie zur Systemverwaltung und für Diskettenzugriffe über INT-Aufrufe (siehe Befehlssatz der CPU 8086) zur Verfügung stehen.

MS-DOS setzt seinerseits voraus, daß der Computer, auf dem es gestartet wird, gewissen Konventionen genügt. Dies betrifft insbesondere die von MS-DOS selbst benötigten hardware-abhängigen Ein- und Ausgabe-Funktionen, die im ROM-BIOS (BIOS = Basic Input/Output System) implementiert sind. Die folgende Tabelle stellt die wichtigsten beim IBM-PC vorhandenen ROM-BIOS-Aufrufe zusammen. Dabei entsprechen die Interrupt-Vektoren INT 0BH...0FH den vom Interrupt-Controller 8259 verwalteten Hardware-Interrupts IRQ 0...7.

INT	Verwendung
00H	Division durch Null (vom 8086 bzw. 8088 erzeugt)
01H	Single-Step-Interrupt
02H	NMI (non maskable interrupt, RAM-Parity-Fehler)
03H	Interrupt für Debugger-Breakpoints
04H	Divisions-Überlauf
05H	Routine zum Ausdrucken des Bildschirms (PrtSc-Taste)
08H	Hardware-Interrupt des Timers 8253 (alle $^1/_{18}$ s)
09H	Hardware-Interrupt der Tastatur (IRQ1)
0BH	Hardware-Interrupt der Schnittstelle COM2
0CH	Hardware-Interrupt des V.24-Bausteins 8250 (IRQ4), ist durch den OUT2-Ausgang des 8250 maskiert (COM1)
0DH	Hardware-Interrupt der Festplatte (IRQ5)
0EH	Floppy-Controller-Hardware-Interrupt (IRQ6)
0FH	Drucker-Interface-Hardware-Interrupt (IRQ7)
10H	Video-I/O: AH=0: Modus in AL einstellen (0...7; siehe SCREEN in GW-Basic) AH=1: Cursor-Art; CH = Startzeile, CL = Endzeile der Punktmatrix

INT	Verwendung
	AH=2: Cursor an Position DL = Spalte, DH = Zeile, BH = Seite stellen
	AH=3: Cursor-Position lesen: DL = Spalte, DH = Zeile, BH = Seite (0 = Grafik), CH = Cursor-Anfangszeile, CL = Cursor-Endzeile der Punktmatrix
	AH=4: Lesen der Lichtgriffel-Position; BH = Seite (0 = Grafik), DL = Spalte, CH = Rasterlinie (0...199), BX = Rasterspalte (0...319/639), AH = Status des Griffelschalters (0 = nicht gedrückt, 1 = gedrückt)
	AH=5: Textseite verändern; AL = Seite (0...7 bei 40 Zeichen, sonst 0...3)
	AH=6: Fenster nach oben scrollen; AL = Zeilen (0 = alle), CL = linker Rand, CH = oberer Rand, DL = rechter Rand, DH = unterer Rand, BH = Attribut für unbenutzte Zeilen
	AH=7: Fenster nach unten scrollen; siehe AH=6
	AH=8: Zeichen (in AL) und Attribut (in AH) des Cursors auf Seite BH ermitteln
	AH=9: Zeichen(folge) ausgeben; CX = Zeichenzahl, BH = Seite, AL = ASCII, BL = Zeichen-Attribut; kein Auto-CRLF am Zeilenende
	AH=A: Zeichen ohne Attribut ausgeben; BH = Bildschirmseite, CX = Zeichenzahl, AL = ASCII-Wert; kein Auto-CRLF am Zeilenende
	AH=B: Farbwerte setzen; AH = Palette, BL = Farbe
	AH=C: Punkt setzen; AL = Farbe (Bit 7=1: XOR), CX = Spalte, DX = Zeile
	AH=D: Farbe eines Punkts in AL einlesen; CX = Spalte, DX = Zeile
	AH=E: Zeichen ausgeben; AL = ASCII, BL = Vordergrundfarbe, BH = Textseite
	AH=F: Video-Status abfragen; Ergebnis: AL = Schirm-Modus, AH = Zeichen pro Zeile, BH = Bildschirmseite
11H	Gerätekonfiguration prüfen (liefert Bitmuster in AL)
12H	Speichergröße ermitteln; AX enthält Größe in KByte
14H	Serielle Schnittstelle; DX=0: COM1; DX=1: COM2 usw.
	AH=0: Parameter setzen; Bits in AL: 7...5: Baudrate (010 = 300 Bd, 100 = 1200 Bd usw.) 4...3: Parität (00 = keine, 01 = ungerade, 11 = gerade) 2: Stopbits (0 = eins, 1 = zwei) 1...0: Wortlänge (10 = 7 Bits, 11 = 8 Bits)
	AH=1: Zeichen in AL ausgeben; Resultat: AH = Line Status
	AH=2: Auf empf. Zeichen warten; Resultat: AL = Zeichen, AH = Line Status (nur Bite 0...3!)
	AH=3: Status abfragen; Resultat: AH = Line Status, AL = Modem-Status (siehe folgende 8250-Beschreibung)

INT	Verwendung
15H	Kassettenrecorder-Ein-/Ausgabe (nur Orig.-IBM-PC)
	AH=0: Recorder-Motor einschalten
	AH=1: Recorder-Motor ausschalten
	AH=2: 256-Byte-Block lesen; ES:BX = Pufferzeiger, CX = Byteanzahl; liefert ES:BX = Ende+1, DX = Zahl wirklich gelesener Bytes, Carry = 1: Fehler mit AH = Fehlercode
	AH=3: Schreiben eines 256-Byte-Blocks mit ES:BX = Pufferadresse, CX = Bytezahl; liefert ES:BX = Ende+1, CX = 0
16H	AH=0: Tastatur lesen; liefert AL = ASCII, AH = Scan-Code
	AH=1: Test, ob Taste gedrückt; Z-Flag = 1: nein, sonst ASCII in AL und Scan-Code in AH
17H	AH=0: Zeichen in AL über parallele Schnittstelle ausgeben; DX = Drucker (0, 1, 2...); liefert AH = 1, wenn Drucker nicht bereit
	AH=1: Druckerport Nr. DX (0, 1...) initialisieren; liefert Status in AH
	AH=2: Drucker-Status abfragen; liefert Status in AH
18H	Zeiger auf Kassetten-Basic (nur Orig.-IBM-PC)
19H	Zeiger auf Bootstrap-Routine
1AH	AH=0: Uhrzeit nach CX:DX lesen; AL = 0, wenn < 24 h
	AH=1: Uhrzeit auf CX:DX stellen
1BH	Zeiger auf Tastatur-Break-Behandlungs-Routine
1CH	Wird 18,2mal pro Sekunde von INT 08H aufgerufen, zeigt auf IRET-Befehl
1DH	Zeiger auf Display-Parameter-Block
1EH	Zeiger auf Disk-Parameter-Block
1FH	Zeiger auf Grafik-Zeichensatz mit Codes 128...255 (wird z.B. von GRAFTABL.COM gesetzt, um deutsche Umlaute in den Grafik-Modi darstellen zu können)

Um ein Programm auf unterschiedlichen MS-DOS-Rechnern lauffähig zu machen, muß es weder umgeschrieben noch angepaßt werden — sofern man sich an die genormten MS-DOS-Systemaufrufe hält. Alle Ein- und Ausgabefunktionen sowie einige Hilfsroutinen sind wiederum über INT-Befehle erreichbar. Die wichtigsten werden im folgenden kurz erläutert:

INT 20H	Programmende, Speicher wird wieder ganz freigegeben
INT 21H	Aufruf aller Betriebssystem-Funktionen (siehe unten)
INT 22H	Zeiger auf Routine, die bei Programmende angesprungen werden soll
INT 23H	Zeiger auf Routine, die bei CTRL-C aufgerufen wird
INT 24H	Zeiger auf Routine zur Behandlung schwerwiegender Fehler (s. u.)
INT 25H	Physikalisches Lesen zusammenhängender Disk-Sektoren
INT 26H	Physikalisches Schreiben zusammenhängender Sektoren
INT 27H	Programmende, Speicher wird bis CS:DX reserviert

Der Aufruf INT 21H kann unterschiedlichste Funktionen auslösen, je nachdem, welcher Wert im CPU-Register AH übergeben wird. Die wichtigsten, am häufigsten gebrauchten davon sind:

00H Programmende
01H Zeichen mit Echo auf Schirm von der Tastatur nach AL holen
02H Zeichen in DL auf Schirm (CON) ausgeben
03H Zeichen von Gerät AUX nach AL einlesen
04H Zeichen in DL nach AUX ausgeben
05H Zeichen in DL an den Drucker ausgeben (PRN)
06H DL=FFH: AL = eingegebenes Zeichen, sonst 0; DL ungleich FFH: Zeichen in DL ausgeben
07H Zeichen von der Tastatur nach AL einlesen, ohne Echo
08H Wie 07H, jedoch Abbruch des Programms mit CTRL-C möglich
09H Durch DX adressierte Zeichenkette bis $-Zeichen ausgeben
0AH Zeichenkette mit Echo bis Return ab Adresse DX einlesen
0BH Eingabestatus; AL = 0: kein Zeichen, AL = FFH: Zeichen im Puffer
0CH Eingabepuffer löschen, dann Funktion in AL aufrufen; zulässig sind die Funktionen 01H, 06H, 07H, 08H und 0AH
0DH Alle offenen Files schließen (reset disk)
0EH Laufwerk in DL voreinstellen (A = 0, B = 1 usw.)
0FH Datei mit DX = File Control Block (FCB) öffnen; o.k., wenn AL = 0
10H Datei mit DX=FCB schließen; o.k., wenn AL = 0
11H Datei mit DX=FCB im Directory suchen; AL = 0, wenn gefunden
12H Nach Funktion 11H nächsten Eintrag suchen (Name z.B. *.BAS)
13H Datei mit DX=FCB löschen
14H Satz aus Datei mit DX=FCB lesen
15H Satz aus Datei mit DX=FCB schreiben
16H Datei mit DX=FCB erzeugen
17H Datei umbenennen
19H Laufwerksnummer abfragen
1AH Disk-Puffer (DTA = Data Transfer Area) setzen
22H Einzelnen Satz schreiben
24H Auswahl eines Satzes für wahlfreien Zugriff
25H INT-Vektor Nr. AL = DS:DX setzen
27H Mehrere Sätze von Disk lesen
28H CX Sätze schreiben, oder, wenn CX=0, Dateigröße ändern
29H Dateinamen bei Adr. SI aufbereiten; AL sollte vorher 0FH sein
2AH Tagesdatum nach CX:DX lesen
2BH Tagesdatum aus CX:DX setzen
2CH Uhrzeit nach CX:DX lesen
2DH Uhrzeit aus CX:DX setzen
2FH Disk-Puffer (DTA) setzen
30H MS-DOS-Versionsnummer holen (0 vor Version 2!)
31H Programmende, Programm bleibt im Speicher
33H Programmabbruch durch CTRL-Break steuern

35H INT-Vektor Nr. AL holen; Ergebnis in ES:BX
36H Disk-Laufwerks-Informationen holen
38H Landesspezifische Informationen holen
39H Sub-Directory anlegen, DX = Name (abgeschlossen mit Null-Byte)
3AH Sub-Directory löschen, DX = Name
3BH Directory wechseln, DX = Pfadname
3CH Datei erzeugen; DX = Name, CX = Attribut (0 = normale Datei), liefert Kanal (Zugriffswort) in AX
3DH Datei öffnen; DX = Name; AL = Zugriffsart (0 = Read, 1 = Write, 2 = R/W), liefert Kanal in AX
3EH Datei schließen, BX = Zugriffswort (Kanal, Handle-Nr.)
3FH Lesen von Gerät/Datei; BX = Zugriffswort, DX = Puffer, CX Bytes
40H Schreiben nach Gerät oder Datei (Reg. wie 3FH)
41H Datei löschen, DX = Name
47H Aktuelles Directory feststellen
48H BX Paragraphen (jeweils 16 Byte) Speicherplatz reservieren
49H Reservierten Speicherplatz wieder freigeben
4AH Ändern der reservierten Speicherplatz-Größe
4EH Directory nach Eintrag durchsuchen
4FH Directory nach nächstem Eintrag durchsuchen
54H Verify (geschriebene Daten prüfen)
56H Datei umbenennen; DX = alter Name, DI = neuer Name
57H Datum und Uhrzeit einer Datei im Directory bearbeiten

Es ist zu erkennen, daß es manche Funktionen scheinbar doppelt gibt. Das liegt daran, weil es CP/M-ähnliche Funktionen gibt (00H bis 012H), die die hierarchische Baumstruktur von MS-DOS nicht unterstützen, und Xenix-ähnliche, die Subdirectories zulassen. Die CP/M-ähnlichen arbeiten mit einem File-Control-Block, der folgendermaßen aufgebaut ist:

Adresse	Inhalt
−7	FFH bei erweitertem FCB
−6...−2	Reserviert
−1	Attribut (MS-DOS-Version 1)
0	Laufwerksnummer
1...8	Filename ohne Extension
9...11	Extension des Filenamens (z.B. COM)
12...13	Aktuelle Blocknummer (high/low)
14...15	Logische Satzlänge (high/low)
16...19	Dateilänge (Bytefolge 3, 4, 1, 2 mit 1 = höherwertiges)
20...21	Datum der Datei-Eröffnung
22...31	Reserviert
32	Aktueller Datensatz im aktuellen Block
33...36	Datensatz-Nummer für wahlfreien Zugriff

Tritt während eines Disk-Zugriffs (oder, bei den Xenix-ähnlichen Funktionen, während eines Zugriffs auf eine Datei oder ein Peripheriegerät) ein schwerwiegen-

der Fehler auf (fatal error), so kümmert sich normalerweise der Kommandoprozessor COMMAND.COM in MS-DOS darum und meldet „Device not ready: abort, retry, ignore?". Von einem Anwenderprogramm aus können solche Fehler über den INT-24H-Vektor abgefangen werden. Die Interrupt-Routine kann in Assemblersprache etwa so aussehen:

```
fatal:  add sp,6    ;IP, CS, Flags vom Stack werfen
        pop ax
        pop bx
        pop cx
        pop dx
        pop si
        pop di      ;restliche Register
        pop bp      ;restaurieren
        pop ds
        pop es
        add sp,4
        popf
        stc
        pushf       ;Carry = Error-Flag setzen
        sub sp,4    ;(wie bei Xenix-Funktionen)
        iret
```

Beim Auftreten eines schweren Fehlers während der Ausführung einer INT-21H-Funktion „landet" diese Routine dann unmittelbar hinter der aufrufenden INT-21H-Anweisung.

Selbstverständlich ist es auch möglich, direkt auf Hardware-Adressen zuzugreifen. Ein Programm, das dies tut, läuft aber dann nicht notwendigerweise auf jedem beliebigen MS-DOS-Rechner, sondern nur auf mehr oder weniger IBM-PC-kompatiblen Geräten.

Trotzdem seien einige oft verwendete I/O-Adressen innerhalb des Kapitels über MS-DOS aufgeführt, da die meisten Hersteller von MS-DOS-Rechnern die Adressenbelegung des IBM-PC weitgehend übernommen haben (lediglich beim Sirius-1, beim Philips-Yes, beim Siemens-PC-D und einigen anderen Geräten ist dies nicht der Fall). Alle I/O-Adressenangaben erfolgen in hexadezimaler Form.

von	bis	Baustein
0000	000F	DMA-Controller 8237
0020	0021	Interrupt-Controller 8259
0040	0043	Timer 8253
0060		Tastatur-Encoder 8041
0061		Systemsteuerung; Bit 1 = Lautsprecher-Ausgang
0062	0067	Systemkonfiguration (DIP-Schalter)
0201		Game-Anschluß (Joystick)
0278	027A	Drucker-Schnittstelle LPT2 (vgl. 03BC!)

02F8	02FE	Serielle Schnittstelle COM2 (optional)
03B4	03B5	Monochrom-Video-Controller 6845; 03B4 = Registeradresse, 03B5 = Daten
03B8		Display-Modus-Steuerung: Bit 1 = Grafikmodus, Bit 3 = Video ein, Bit 5 = blinken, Bit 7 = Seite
03BA		Video-Status; Bit 0 = Sync-Phase, Bit 3 = Punkt an, Bit 7 = gerade kein Vertikal-Sync
03BC (oder 0378)	03BE (oder 037A)	Drucker LPT1. 03BC = Datenport; 03BD = Drucker-Status (LPT1); Bit 3 = Error, Bit 4 = selektiert; Bit 5 = Papierende, Bit 6 = Acknowledge, Bit 7 = Busy; 03BE = Drucker-Steuerung (LPT1); Bit 0 = Strobe, Bit 1 = Auto-LF, Bit 2 = Init, Bit 3 = Select, Bit 4 = IRQ7-Freigabe, wenn Drucker bereit
03D4	03D5	Farbgrafik-Video-Controller 6845; 03D4 = Registeradresse, 03D5 = Daten
03F2	03F5	Floppy-Controller NEC 765
03F8	03FE	Serielle Schnittstelle COM1 (8250); s. u.

Das mit dem Baustein 8250 von National Semiconductor realisierte V.24-Interface ist einer genaueren Erläuterung wert. Die von COM1 belegten Adressen haben folgende Bedeutung, wobei DLAB das Bit 7 an der Adresse 03FB ist (Divisor Latch Access Bit):

03F8:	Sende-/Empfangsdaten (DLAB = 0); Baudrate Low (DLAB = 1)
03F9:	IRQ4-Freigabe (DLAB = 0); Baudrate High (DLAB = 1); Interrupt-Bits: 0 = Daten empfangen, 1 = Senderegister leer, 2 = Empfangsstatus, 3 = Modemstatus
03FA:	Interrupt-Flagregister-Bits: 0 = Interrupt steht an, 1 und 2 = Code für Interruptquelle: 11 = Empfangsstatus, 10 = Daten empfangen, 01 = Senderegister leer, 00 = Modemstatus
03FB:	Parameter-Bits: 0 und 1 = Wortlänge (5, 6, 7, 8 Bits), 2 = Stopbits (1, 2), 3 = Parity ein, 4 = gerade Parity, 5 = festes Paritätsbit, 6 = Break senden, 7 = DLAB
03FC:	Modemsteuerungs-Bits: 0 = DTR, 1 = RTS, 2 = Out1, 3 = Out2 (maskiert IRQ4!), 4 = Loop (interne Schleife für Tests)
03FD:	Empfangsstatus-Bits (Line Status): 0 = Daten empfangen, 1 = Overrun, 2 = Parity-Error, 3 = Framing-Error, 4 = Break empfangen, 5 = Senderegister leer, 6 = Sende-Schieberegister leer
03FE:	Modemstatus-Bits: 0 = CTS hat sich geändert, 1 = DSR hat sich geändert, 2 = RI-Flanke, 3 = CD hat sich geändert, 4 = CTS, 5 = DSR, 6 = RI (Telefon klingelt!), 7 = CD (Träger)

Das ROM-BIOS greift mit INT 14H auf diese Adressen zu. Von seiner Verwendung ist im allgemeinen abzuraten, da mangels Empfangspuffer bei höheren Baudraten leicht Zeichen verloren gehen können.

Um eine selbst geschriebene Interrupt-Routine zu aktivieren, die empfangene Zeichen vom 8250 abholt und einem Ringpuffer übergibt, muß man nacheinander folgendes tun:

1. Baudrate einstellen (z.B. mit dem Mode-Befehl in MS-DOS).
2. Interrupt-Vektor 0CH (IRQ4) auf Interrupt-Routine zeigen lassen.
3. Adresse 03F8 einmal lesen, um das Interrupt-Bit für „Daten empfangen" zu löschen.
4. Im Interrupt-Controller 8259 den IRQ4 freigeben (Bit 4 an Adresse 0021 mit AND 0EFH auf Null rücksetzen).
5. Bit 0 an der Adresse 03F9 und Bit 3 bei 03FC auf 1 setzen, um den Interrupt bei empfangenen Daten freizugeben.

Beim nächsten empfangenen Zeichen wird nun die Interrupt-Routine angesprungen, die den Zeichencode in einen Ringpuffer ablegen kann, von wo aus das (langsamere) Hauptprogramm ihn gemütlich bei nächster Gelegenheit abholen kann. Vor dem Ende der Interrupt-Routine ist das Übergeben des Befehlsworts 64H an die Adresse 0020 nötig (Specific End of Interrupt 4).

Gerade dieses Beispiel ist typisch dafür, daß man trotz der vielen vom Betriebssystem zur Verfügung gestellten Möglichkeiten um eine hardwarenahe Systemprogrammierung nicht herumkommt.

4.1.3 Unix

Das Betriebssystem Unix wurde von den amerikanischen Bell Laboratories entwickelt, die zum Informationstechnik-Konzern AT+T (American Telegraph and Telephone) gehören. Zunächst diente es der Unterstützung der Software-Entwicklung auf einem DEC-Minicomputer PDP-7, später (1973) auf dem Nachfolgetyp PDP-11. Da man bei den Bell Labs erkannte, daß es mühsam und teuer sein würde, das Betriebssystem regelmäßig auf neue Computertypen umzuschreiben, entstand 1973 die zweite Unix-Version gleich zu 90% in maschinen-unabhängigem Code, nämlich in der Sprache C. Heute ist Unix in mehreren Varianten auch auf den „gehobenen" 16-Bit-Mikrocomputern zu finden, wie z.B. PCS-Cadmus, IBM-AT, Fortune 32:16 und anderen.

Unix ist ein Multi-User-System, wobei bis zu 128 Benutzer quasi gleichzeitig im Time-Sharing-Betrieb (Zeitscheiben-Betrieb) mit einem zentralen Unix-Rechner arbeiten können. Allerdings sollte man sich bezüglich dieser Zahl keinen Illusionen hingeben: Bei Mikrocomputern ist normalerweise schon bei vier bis acht Benutzern die Systembelastung so hoch, daß sich unzumutbare Antwortzeiten ergeben können. Da zu Unix eine Vielzahl von Routinen und Utility-Dateien gehört, ist für den Betrieb eine Festplatte sowie ein Speicher von wenigstens 512 KByte nahezu unbedingte Voraussetzung. Ebenso ist ein Prozessor erforderlich, der virtuelle Adressierung unterstützt (z.B. 68020), oder, falls dies nicht der Fall ist, entsprechende MMU-Bausteine (Memory Managing Unit).

Wegen dieser vielfältigen Anforderungen an die Computer-Hardware kommt Unix konzeptionell in erster Linie für Rechner an der Grenze zwischen Mikro-

und Minicomputern in Betracht. Es ist insbesondere nicht zu erwarten, daß es die typischen Ein-Benutzer-Systeme wie CP/M oder MS-DOS verdrängen wird, insbesondere nachdem speziell MS-DOS ebenfalls bereits eine Reihe von Features bietet, die bislang Unix vorbehalten waren (Filter, Pipes, hierarchische Directory-Bäume).

Grundsätzlich erfolgen unter Unix alle Operationen Datei-orientiert. Das bedeutet, daß es praktisch ausschließlich „externe" Kommandos gibt, die durch Laden und Starten der dazugehörigen Routine ausgeführt werden, und daß alle Ein- und Ausgabe-Vorgänge über Dateien als Puffer laufen. Dies bedeutet aber auch, daß Unix nicht als Echtzeit-Betriebssystem geeignet ist: Wenn z.B. Daten auf einen Drucker ausgegeben werden sollen, dann geschieht das unter Umständen nicht augenblicklich nach Eingabe des entsprechenden Kommandos, sondern erst dann, wenn das System Zeit dafür hat, die Zwischendatei auszugeben.

Die Benutzer-Schnittstelle (d.h. die Kommando-Sprache und die System-Meldungen) wird durch die sogenannte „Shell" dargestellt, die wiederum aus ASCII-Dateien besteht. Wie in MS-DOS kann man Dateien über Filter in andere schleusen (Sortieren usw.), Eingaben mit „<" bzw. Ausgaben mit „>" umleiten (mit „>!" auch auf schon bestehende Dateien) und Parameter von einem Kommando zum nächsten durch „Pipelining" (senkrechter Strich wie in MS-DOS!) übergeben. Wenn in diesem Kapitel von Directory-Pfaden, dem aktuellen Directory, von Filtern und Pipes die Rede ist, so sei auf den Anfang des Kapitels über MS-DOS verwiesen.

Unix-Standardkommandos

Unix besitzt eine Vielzahl von Kommandos und ergänzenden Routinen. Hier seien deshalb nur die wichtigsten aufgeführt, ohne den Anspruch auf Vollständigkeit erheben zu wollen. Immerhin bekommt man ein Gefühl für die Mächtigkeit des Unix-Befehlsumfangs. Es ist übrigens nicht gleichgültig, ob man Kommandos mit Groß- oder Kleinbuchstaben schreibt; diese haben z.T. unterschiedliche Wirkungen (siehe z.B. ls).

apropos
Suche in der Datei /usr/whatis, die alle Unix-Programme kurz beschreibt, nach einem Stichwort und ggf. Ausgabe der entsprechenden Zeilen.

at
Auftrag an den Rechner, ein Shell-Script (vgl. script) zu einer bestimmten Zeit zu starten. Allerdings wird die zugehörige Datei /usr/lib/crontab normalerweise nur alle fünf Minuten abgefragt, so daß kürzere Intervalle nicht sinnvoll sind, es sei denn, man ändert von Hand diese Datei.

banner
Schreibt Worte in Großschrift, indem jeder Buchstabe im Format 6 × 6 aus dem Zeichen „#" (Doppelkreuz) zusammengesetzt wird (ideal für Überschriften in meterlangen Listings!).

bc
Erlaubt einfache arithmetische Berechnungen mit Variablennamen aus Kleinbuchstaben. Die gewünschte Stellenzahl kann man mit scale vorher vereinbaren.

cal
Liefert einen Monats- (cal August 1985) oder Jahreskalender (cal 1999).

calendar
Gestattet die automatische Terminkalender-Führung für alle Benutzer. Im aktuellen Directory des Benutzers wird nachgesehen, ob die Datei „calendar" Zeilen mit dem heutigen oder morgigen Datum enthält; wenn ja, werden diese ausgegeben. Das Datum wird im amerikanischen Format geführt (Monat, Tag, Jahr).

cat
Gibt den Inhalt einer Datei ohne Unterbrechung auf den Bildschirm aus (vgl. dog).

cd
Wechsel des aktuellen Directory. Mit „.." kann dabei das vorhergehende Directory angewählt werden, andernfalls ist der vollständige Pfad anzugeben.

cmp
Vergleich zweier Dateien. Beim ersten nicht übereinstimmenden Zeichen werden Zeile und Zeichenposition ausgegeben.

comm
Vergleich zweier Dateien. Das Ergebnis wird in drei Spalten geliefert: Die erste enthält die Nummern der Zeilen, die nur in der ersten Datei vorkommen, die zweite jene, die nur in der zweiten Datei enthalten sind, und die dritte nur die, die in beiden Dateien vorkommen.

cp
Kopieren einer Datei, d.h. Erzeugen eines Duplikats mit anderen Namen im selben oder in einem anderen Directory.

crypt
Ver- und Entschlüsselung von Daten mit dem nach crypt folgenden Codewort durch bitweise Exklusiv-Oder-Verknüpfung einer Datei mit der Codewort-Zeichenfolge. (Die ursprüngliche Datei sollte man dann zweckmäßig löschen!)

csh
Anwahl der sogenannten C-Shell, eines Interpreters, der Ähnlichkeiten mit der Sprache C aufweist. Damit ist es möglich, sich selbst neue Unix-Befehle zu definieren. Eingegebene C-Shell-Befehle werden automatisch in eine History-Liste geschrieben, so daß der Arbeitsablauf später rekonstruierbar ist. Typische C-Shell-Befehle sind: alias, alloc, break, case, chdir, continue, echo, else, end, endif, exit, foreach, goto, history, if, nice, nohup, onintr, repeat, set, shift, source, switch, then time, umask, wait, while und andere. (Es wird empfohlen, hierzu Spezialliteratur zu konsultieren.)

csplit
Ähnlich split, die Trennung der Dateien kann man jedoch durch Zeichenfolgen (zwischen Schrägstrichen, z.B. /text/) oder Zeilennummern angeben. Es werden so viele neue Dateien plus eine erzeugt, wie Argumente (z.B. Zeilennummern) vorhanden sind.

cu
Utility zum Transfer von Dateien (die keinen Objektcode enthalten dürfen!) vom einen Rechner auf einen anderen.

date
Anzeige von Datum, Wochentag und Uhrzeit im amerikanischen Format.

daytime
Liefert die Uhrzeit in englischem Klartext, z.B. „quarter to ten".

dc
Ähnlich bc, jedoch wird hier ein Rechner mit Stack und Umgekehrter Polnischer Notation simuliert.

diff
Vergleich zweier Dateien; Zeilen, die nur in Datei 1 vorkommen, werden mit einem Kleiner-Zeichen markiert, jene, die nur in Datei 2 enthalten sind, mit einem Größer-Zeichen.

diff3
Funktion ähnlich diffmk (s.u.), jedoch ist es möglich, drei Dateien miteinander zu vergleichen.

diffmk
Vergleich zweier Dateien. Es wird eine zusätzliche Datei erzeugt, die die Zeilen von Datei 2 enthält, wobei alle jene, die nicht mit denen von Datei 1 identisch sind, mit „.mc" markiert werden.

dog
Wie cat, jedoch werden immer nur 23 Zeilen ausgegeben. Zum Weiterblättern ist Return zu drücken.

echo
Zeichenfolgen oder Variablen-Inhalte auf die Standard-Ausgabe schreiben.

echo2
Wie echo, allerdings wird auf die Standard-Fehlerausgabe geschrieben.

ed
Datei editieren. Nach ed ist ein Dateiname zu nennen. Folgende Unterkommandos gibt es:

.	Texteingabe beenden (muß am Zeilenanfang stehen)
a	Text hinter die letzte Zeile anfügen
c	Zeile(n) ändern
d	Angegebene Zeile(n) löschen
j	Mehrere Zeilen zu einer verbinden
l	Text listen (auch Steuerzeichen)
m	Verschieben von Zeilen (z.B. 6,9m23)
p	Ausgabe auf Bildschirm
q	Editor verlassen
s	Zeichen ersetzen, z.B. s/alt/neu/
t	Kopieren von Zeilen (z.B. 6,9t23)
u	Letzten Befehl rückgängig machen
w	Datei abspeichern

expand
Wandelt in einer Datei Tabulator-Zeichen in die entsprechende Zahl von Leerräumen um.

file
Liefert den Typ einer Datei, z.B. „archive", „ascii", „directory" und anderes mehr.

find
Suche nach Dateien (Syntax: find Anfangs-Directory Suchbegriff). Sehr hilfreich bei umfangreichen Directories; typische Eingabe zum Suchen des Programms ADVENTURE ab dem vorhergehenden Directory:

```
find .. -name ADVENTURE -print
```

getname
Ausgabe der ersten Zeilen von Manual-Dateien auf die Standard-Ausgabe, z.B. um die Einträge für whatis zu erzeugen.

gets
Liest eine mit Return beendete Zeile von der Stadard-Eingabe und schreibt sie auf die Standard-Ausgabe.

grep
Dient zum Auffinden von Zeichenfolgen in Dateien. Normalerweise werden alle Zeilen ausgegeben, in der die Zeichenfolge vorkommt, mit der Option −1 dagegen die Zeilen, in der sie nicht vorkommt. Mit −n kann man jeder Zeile eine Nummer voranstellen lassen (vgl. FIND in MS-DOS!). Sollten Zeichen wie Stern, Dollar, runde und eckige Klammern, Pfeil nach oben (Potenzpfeil) und Backslash (nach links geneigter Schrägstrich) vorkommen, sollte man den ganzen String in Anführungszeichen setzen, um falsche Interpretationen durch die Shell zu vermeiden.

head
Gibt die ersten zehn Zeilen einer Datei auf die Standardausgabe aus. Mit −n (n = ganze Zahl) kann man die Zeilenzahl selbst bestimmen, z.B. head −25 Name.

id
Liefert die Benutzernummer und den dazugehörigen Namen, mit dem das Terminal „eingeloggt" wurde.

line
Identisch mit gets, siehe dort.

ln
Erzeugt einen „Link": Eine Datei wird scheinbar kopiert, so daß sie mit unterschiedlichen Namen oder Directories zweimal vorhanden ist, ohne daß sie jedoch wirklich dupliziert wird.

look
Sucht in einer Datei nach Zeilen, die mit einer bestimmten Zeichenfolge beginnen; die Syntax lautet look Zeichenfolge Dateiname.

lpr
Spooler-Funktion zur Ausgabe von Dateien auf den Drucker (Line Printer). Das geschieht im Hintergrund; dem Benutzer wird jedoch Meldung erstattet, wenn der gewünschte Ausdruck fertig ist.

ls
Auflisten einer Datei; falls keine angegeben ist, wird das aktuelle Directory angezeigt. Mögliche Optionen sind:

−l	Langform des Listings mit Datum der letzten Änderung, Angabe von Zugriffs-Beschränkungen usw.
−a	Auch alle Dateien, die mit einem Punkt beginnen und sonst nur für „Superuser" zugänglich sind, werden gelistet
−d	Nur der Name eines Subdirectory wird ausgegeben, nicht aber sein Inhalt
−D	Es werden nur Subdirectories gelistet

Beispiel: ls −a .geheim

man
Abkürzung für „Manual" (Handbuch). Das in Englisch geschriebene Unix-Betriebshandbuch ist in weiten Teilen von der Festplatte abrufbar.

mail
Programm zum Nachrichtenaustausch mit anderen Benutzern des Systems:

mail	listet empfangene Nachrichten
mail hans	erlaubt die Eingabe einer Nachricht an den Benutzer Hans; sie muß mit Return und CTRL-D beendet werden

mesg
Mit mesg −y kann man den Empfang von Nachrichten anderer Benutzer zulassen, mit mesg −n dagegen sperren.

mkdir
Erzeugen eines neuen Directory (wie in MS-DOS); vor dem ersten Anlegen einer Datei des neuen Directory erforderlich.

more
Seitenweise Ausgabe einer Datei auf den Bildschirm (je 23 Zeilen); weiterblättern ist mit der Leertaste möglich, während Return nur die nächste Zeile ausgibt. Zusätzlich stehen z. B. Befehle zum Suchen von Zeichenfolgen zur Verfügung; mit h erhält man eine entsprechende Liste.

mv
Umbenennen einer Datei in einem Directory oder Transfer in ein anderes Directory. Die ursprüngliche Datei wird dabei im Gegensatz zu cp gelöscht.

newgrp
Mit Hilfe eines zusätzlichen Paßworts (wie in der Datei /etc/group) kann man so seine Zugehörigkeit zu einer bestimmten Benutzergruppe ändern.

news
Hausinterner Informationsdienst. Jede Meldung ist als eigene Datei im Directory /usr/lib/news gespeichert; mit der Option −n erhält man eine Übersicht.

nice
Erniedrigen der Priorität eines Programms, z. B. um zwei Stufen mit nice −2. Nur der Superuser kann die Priorität auch erhöhen, z. B. mit nice − −3.

nl
Gibt eine Datei mit Zeilennummern aus.

nohup
Sperrt die Unterbrechung des angegebenen Programms durch Interrupts.

pr
Formatierte Ausgabe einer Textdatei mit dem Dateinamen oder, mit der Option −h, eines beliebigen Textes als Überschrift. Mit −n (n = ganze Zahl) erfolgt die Ausgabe mit n Zeichen Breite, mit +n beginnt sie erst bei der Seite n. Mit −m kann man mehrere Dateien nebeneinander (jede in einer eigenen Spalte) ausgeben, wobei sich die Spaltenbreite aus 72 dividiert durch die Zahl der Dateien ergibt.

printenv
Liefert die in der Shell gesetzten Umgebungs-Variablen.

pwd
Anzeige des aktuellen Directory mit dem dazugehörigen Pfad.

reset
Rücksetzen der Terminal-Parameter auf den Zustand, wie man sie beim „Einloggen" normalerweise vorfindet.

rm
Löschen einer Datei oder, mit der Option −r, auch des Directory. Mit −i wird man jedesmal erst gefragt, ob man die Datei wirklich löschen möchte. Mit −f wird eine Fehlermeldung vermieden, falls die Datei nicht existiert.

rmdir
Löschen eines Directory; funktioniert jedoch nur dann, wenn dieses Directory keine Dateien enthält.

sed
Stream-Editor; dient zum Bearbeiten von Dateien, indem man sie sozusagen durch den Editor „fließen" läßt.

script
Nach script werden alle Eingaben des Benutzers auf einer eigenen Datei mitprotokolliert, die entweder angegeben wird oder „typescript" heißt (funktioniert nicht bei Programmen, die mit CTRL-D beendet werden!).

sdiff
Wie diff (s.d.); das Ergebnis wird auf die Standard-Ausgabe geschrieben und alle Zeilen mit einem senkrechten Strich für unterschiedliche, mit einem Leerzeichen für gleiche und mit > bzw. < für Zeilen, die nur in einer der beiden Dateien vorkommen, markiert.

sh
Interpreter-Programm namens Shell, das folgende Befehle versteht: break, case, continue, cd, do, done, elif, else, esac, eval, exec, exit, export, fi, if, in, newgrp, read, readonly, set, shift, test, then, times, trap, umask, wait, while. Eine Weiterentwicklung stellt die C-Shell dar (siehe csh), die noch weit mehr Befehle besitzt.

sleep
Der Rechner wartet mit der Befehlsausführung um den nach sleep angegebenen Sekundenwert.

sort
Sortieren von Dateien. Optional kann man mit −t veranlassen, daß das erste relevante Zeichen in jeder Zeile der Datei das nach dem mit −t vereinbarten Trennsymbol stehende Zeichen ist. Mit −n sorgt man dafür, daß Zahlen unterschiedlicher Länge richtig sortiert werden können (z.B. ist 12 größer als 9, würde bei ASCII-Sortierprogrammen jedoch falsch eingeordnet).

spell
Untersucht eine in englischer Sprache geschriebene Textdatei auf eventuelle Schreibfehler mit Hilfe des Referenz-Dictionary /usr/dict.

split
Aufspalten einer Datei in bis zu 676 kleinere. Wird nur eine Datei nach Split genannt, enthalten die kleineren Dateien je 1000 Zeilen; optional kann man hinter split jedoch auch die gewünschte Zeilenzahl angeben, z.B. split −30 name. Schreibt man split −400c name, so enthält jede der erzeugten Dateien 400 Zeichen. Die erzeugten Dateien erhalten als Namen den ursprünglichen mit zwei zusätzlichen Buchstaben (aa...zz) oder, wenn kein Name angegeben wurde, ein x und weitere zwei Buchstaben.

su
Mit dem richtigen Paßwort kann man sich durch su zum „Superuser" machen, so daß man alle Dateien und Directories lesen und ändern kann.

tail
Gibt die letzten zehn Zeilen einer Datei auf die Standardausgabe aus. Mit −n kann man wie beim Kommando head die Zeilenzahl auch selbst bestimmen.

talk
Nach talk Benutzer kann man einem anderen Rechnerbenutzer zeichenweise Meldungen auf den Schirm schreiben und umgekehrt; beenden mit CTRL-D.

tee
Sorgt dafür, daß eine Ausgabe gleichzeitig auf die Standard-Ausgabe und auf die nach tee angegebene Datei erfolgt, z. B. bei Pipelines.

touch
Setzt die Zeitangabe, wann eine Datei zum letzten Mal geändert wurde, auf das aktuelle Datum und die momentane Uhrzeit, z. B. touch Name.

tty
Liefert die Bezeichnung des Terminals bzw. der Rechner-Schnittstelle, an der man gerade arbeitet.

uniq
Ausgabe einer Datei, wobei alle Zeilen, die mit der vorherigen Zeile identisch sind, unterdrückt werden.

vi
Komfortablerer Editor als ed, erst ab Unix Version V von Bell implementiert; arbeitet bildschirm-orientiert.

vls
Ähnlich ls, es ist aber auch möglich, durch Cursor-Bewegungen das vorhergehende oder nachfolgende Directory zu listen. Directories werden mit einem Schrägstrich, ausführbare Dateien mit einem Stern markiert. Mit Taste L kann man die lange Listing-Form erhalten (vgl. ls). Drückt man das Fragezeichen und Return, erhält man eine Befehlsübersicht des vls-Programms.

vnews
Wie news, jedoch erhält man (ähnlich vls) eine Übersicht der Dateien im Directory /usr/lib/news und kann mit dem Cursor die gewünschte auswählen.

wc
Ermittelt den Umfang einer Datei und besitzt folgende Optionen:

−c	Zahl der Zeichen (characters)
−w	Zahl der Worte (words)
−l	Zahl der Zeilen (lines)

whatis
Erlaubt den schnellen Zugriff auf die Handbuch-Datei, um sich über Systemprogramme zu informieren. Nach whatis ist daher der Name des jeweiligen Programms zu nennen.

whereis
Liefert eine Information darüber, wo ein bestimmtes Programm sowie die zugehörige Beschreibung zu finden ist, vorausgesetzt, es steht in einem der Directories /bin, /usr/bin oder /etc.

who
Liefert die Namen aller momentan am Rechner arbeitenden Benutzer und die zugehörigen Terminal-Bezeichnungen.

whoami
Liefert den Benutzernamen, unter dem das Terminal „eingeloggt" wurde, sowie Informationen über die zugehörigen Zugangs-Erlaubnisse.

whodo
Anzeige aller momentan am Rechner arbeitenden Benutzer und ihrer Aktivitäten.

whois
Sucht in der Datei /etc/whois nach dem angegebenen Benutzernamen und liefert ggf. den Eintrag auf dem Bildschirm.

write
Nach write Benutzer kann man einem anderen Rechner-Benutzer zeilenweise Meldungen auf den Bildschirm schreiben; beenden mit CTRL-D.

Die Unix-Shell

Die unter dem Betriebssystem Unix zur Verfügung stehende Benutzer-Schnittstelle namens „Shell" enthält einen Interpreter für eine spezielle Programmiersprache (sie ist absichtlich hier und nicht im Kapitel über Sprachen aufgeführt, da sie nur unter Unix verwendet wird; eine Erweiterung der Shell stellt die C-Shell dar). Bedingt durch das Interpretieren läßt sich keine allzu hohe Geschwindigkeit erzielen. Die Programme können in Dateien abgespeichert werden und müssen dann als solche markiert werden (z.B. chmod 711 NAME). Wenn man ein Shell-Programm erstellt, geht man normalerweise so vor:

vi NAME	Aufruf des Editors vi zum Erstellen der Datei NAME
...	Eingabe des Shell-Programms
:wq	Programm abspeichern und Editor verlassen
chmod 711 NAME	Datei als Programm kennzeichnen
mv NAME /usr/bin/NAME	Datei ins überall zugängliche Directory transportieren

logout	System verlassen
login	Wieder eintreten
NAME	Aufruf und Ausführen der Befehle in NAME

Variablen können Zahlen oder Zeichenfolgen (Strings) aufnehmen und werden durch ein vorangestelltes Dollarzeichen gekennzeichnet. Folgt dem Dollarsymbol eine Zahl, so handelt es sich um eine sogenannte Positionsvariable, die der Reihe nach eingegebene Argumente übernimmt (vgl. die substituierten Werte %1, %2 usw. in MS-DOS!). $§ faßt alle Positions-Variablen zu einer einzigen zusammen. In der Shell ist eine Reihe von Variablen für bestimmte Zwecke vordefiniert:

$!	Nummer des letzten aufgerufenen Hintergrund-Befehls
$#	Nummer des laufenden Shell-Multitasking-Prozessors
$?	Exit-Code des letzten Befehls (0 = wahr bzw. kein Fehler)
$−	Liefert die gesetzten Flags (s. u.)
$MAIL	Name der Datei für Mitteilungen der Benutzer
$PATH	Directory-Liste mit Unix-Kommandos (meist /usr/bin, /etc/bin)
$PS1	Shell-Prompting-Zeichen (meist $)
$PS2	Prompt für weitere Eingaben (meist >)
$IFS	Leerraum-Codierung

Darüber hinaus gibt es eine Reihe von Flags, die bestimmte Status-Meldungen übergeben und mit set gezielt gesetzt werden können (0 = wahr), auch mehrere mit einer Anweisung, z. B. set −vx. Folgende Flags gibt es u. a.:

−e	Ausgangsstatus eines Programms; 0 = wahr bzw. kein Fehler
−i	Null, wenn die Shell interaktiv ist (I/O-Shell, meist 0)
−n	Wenn Null, werden Befehle zwar gelesen, aber nicht ausgeführt
−t	Nur einen einzigen Befehl ausführen, dann Abbruch
−u	Fehlermeldung beim Aufruf nicht gesetzter Variablen
−v	Beim Einlesen von Befehlen werden diese auch ausgegeben
−x	Beim Ausführen von Befehlen werden diese auch ausgegeben
−	Rücksetzen der Flags −v und −x

Als Befehle stehen in der Shell eine Reihe neuer Abkürzungen zur Verfügung. Man muß zugeben, daß Shell-Programme auf den ersten Blick recht schwer verständlich sind; andererseits eignet sich die Shell durchaus auch für komplexere Aufgaben:

:
Ein Doppelpunkt und ein unmittelbar darauffolgender Leerraum teilt der Shell mit, daß bis zum Zeilenende ein nicht zu übersetzender Kommentar folgt.

break
Gewaltsames Verlassen einer for- oder while-Schleife. Wenn man mehrere verschachtelte Schleifen-Ebenen durchspringen will, kann man die Ebenen als Zahl hinter break angeben.

case...in
Vergleich eines Werts mit einer Liste und Ausführung zugeordneter Befehle (vgl. Pascal!). Beispiel:

```
case $aa in
1)    echo $aa;;
A*)   echo $aa;;
*)    echo „Eingabe ist weder 1 noch beginnt sie mit A!"
```

continue
Statt den nächsten Befehl in einer Schleife auszuführen, wird lediglich das nächste Argument (z.B. die nächste Positionsvariable) geholt.

eval Arg1 Arg2 usw.
Die nach eval geschriebenen Argumente werden als Shell-Eingabe übernommen und die Befehle ausgeführt.

exec Arg1 Arg2 usw.
Führt die Befehle in Arg1 aus, ohne einen neuen Prozeß zu generieren. Das bedeutet, daß in späteren Zeilen stehende Befehle nicht mehr ausgeführt werden, sondern statt dessen jene in Arg1. Die Shell hat also das laufende Programm durch Arg1 ersetzt. Die Parameter Arg2 usw. werden an Arg1 übergeben.

exit n
Beenden eines laufenden Shell-Programms. Dabei wird die Zahl n als Ausgangs-Parameter übergeben.

export Var
Ruft das laufende Shell-Programm weitere auf, so kann dabei Var als Parameter-Übergabe-Variable verwendet werden.

for...in...do...done
Schleifen-Konstruktion. Die korrekte Syntax muß lauten:

```
for Schleifenvariable
in Wiederholungszahl
do Befehle
done
```

if...then...else...fi
Bedingte Anweisung, die stets mit if beginnen und mit fi enden muß:

```
if Bedingung
then Anweisung, wenn Bedingung wahr
else Anweisung, wenn Bedingung falsch
fi
```

read Var1 Var2 usw.
Liest eine Zeile (!) vom Terminal ein und ordnet die einzelnen Wörter darin den Variablen Var1, Var2 usw. zu.

readonly Var1 Var2 usw.
Sperrt die Variablen Var1, Var2 usw. vor unbeabsichtigtem Überschreiben; auf sie kann nur noch lesend zugegriffen werden.

set
Listet alle definierten Variablennamen und deren Werte. Stellt man einen Strich und einen Flag-Buchstaben hinter set (z.B. set −e), wird das entsprechende Flag gesetzt. Folgende Argumente ohne Strich, werden sie der Reihe nach den sog. Positionsvariablen ($1, $2 usw.) zugewiesen.

shift
Löscht die Positionsvariable $1, indem $2 zu $1 wird, $3 zu $2 usw. (vgl. MS-DOS!). $0 bleibt hierbei jedoch unverändert.

test
Liefert 0, wenn eine der folgenden Optionsangaben, denen ein Dateiname folgen muß, wahr ist:

−d	Datei existiert und ist Directory
−f	Datei existiert, ist jedoch kein Directory
−r	Datei existiert und ist lesbar
−s	Datei existiert und ist nicht leer
−w	Datei existiert und kann beschrieben werden

Zusätzlich gibt es noch folgende Optionen nach test:

−n S	S ist kein Leerstring
−t i	Die Datei mit der Indexzahl i ist gerade in Bearbeitung
−z S	S ist ein Leerstring (Länge = 0)
S	S ist kein Leerstring
S1 = S2	Die Strings S1 und S2 sind gleich
S1 != S2	Die Strings S1 und S2 sind ungleich
I −eq J	Die (ganzen) Zahlen I und J sind gleich
I −gt J	I ist größer als J
I −ge J	I ist größer oder gleich J
I −ne J	I ist ungleich J
I −le J	I ist kleiner oder gleich J
I −lt J	I ist kleiner als J

trap Befehl n
Der nach trap genannte Befehl wird beim Verlassen des Programms ausgeführt, wenn die Zahl n Null ist, andernfalls, wenn das Signal n (entspricht einem Interrupt) empfangen wird. Fehlt die Angabe eines Befehls, so werden alle traps mit der entsprechenden Signalnummer gelöscht.

umask ooo
Nach umask muß eine Oktalzahl mit maximal drei Ziffern stehen, die die Zugriffs-Erlaubnis neu erstellter Dateien steuert (nicht immer implementiert).

wait n
Wartet, bis der Prozeß mit der Nummer n beendet ist und meldet dessen Ausgangs-Status (vgl. exit n).

while...do...done
Schleife, die solange durchlaufen wird, bis eine Bedingung nicht mehr wahr ist. Syntax:

a) while Befehl ; done
   Abbruch, wenn „Befehl" nicht mehr ausführbar ist, z. B. keine Argumente mehr vorhanden
b) while Bedingung ; do Befehl ; done
   Abbruch, wenn „Bedingung" nicht Null liefert (vgl. test)

4.1.4 Sonstige Standard-Betriebssysteme

Nur wenige herstellerunabhängige Betriebssysteme konnten sich bisher einer weiteren Verbreitung erfreuen. Darüber hinaus gibt es eine ganze Menge, die für besondere Anwendungsfälle konzipiert wurden und in gewissen Bereichen durchaus ihre Berechtigung haben. Einige von ihnen seien deshalb im folgenden kurz besprochen, ohne auf ihren Kommando-Umfang im einzelnen einzugehen. Man muß sich allerdings darüber in klaren sein, daß man sich bei ihnen im Vergleich zu den bisher genannten ungleich schwerer tut, im Notfall Rat, Unterstützung und vor allem Anwender-Software zu bekommen.

Eumel

Das Multiuser-System Eumel wurde vom Hochschul-Rechenzentrum der Universität Bielefeld in Zusammenarbeit mit der Gesellschaft für Mathematik und Datenverarbeitung (GMD) entwickelt. Wie bei Unix ist eine virtuelle Speicherverwaltung implementiert, so daß Programme und Datensätze länger als der Arbeitsspeicher werden dürfen, indem Platten- bzw. Floppy-Dateien als Bestandteil des Speichers betrachtet werden. Zu Eumel gehört die Pascal-ähnliche Programmiersprache Elan. Bisher wird Eumel vorwiegend als Informatik-Lernhilfe und als Hilfsmittel in einigen Schulverwaltungen eingesetzt; eine Mikrocomputer-Implementation wurde u. a. von Olivetti für den Computer M-20 realisiert.

Flex

Das Single-User-System Flex wurde für die 8-Bit-Prozessoren 6800 und 6809 entwickelt. Wie in OS-9 werden eine Reihe von Unix-ähnlichen Eigenschaften realisiert, der Systemkern ist allerdings wesentlich umfangreicher und komfortabler als bei OS-9. Trotz einiger Versuche, es auch auf 6502-Rechnern laufen zu lassen, konnte sich Flex nie so recht durchsetzen.

iRMX

Der Prozessor-Hersteller Intel bietet dieses Multitasking-Betriebssystem für die CPUs der 8086-Familie an. Wie OS-9 ist iRMX für Echtzeit-Aufgaben geeignet, wird bisher allerdings recht selten und ausschließlich im industriellen Bereich eingesetzt.

MSX

MSX heißt ein 1982 von Microsoft entwickeltes Betriebssystem für den Mikroprozessor Z80, das sich an den Erfordernissen von Heimcomputern orientiert und bisher vor allem von japanischen Herstellern eingesetzt wurde, aber auch z. B. von Philips (1984). Grundgedanke war, die Austauschbarkeit von Programmen für Heimcomputer unterschiedlicher Hersteller zu erreichen. Im Gegensatz zu den übrigen hier erwähnten Betriebssystemen werden als Massenspeicher-Medium nicht Disketten, sondern Tonband-Kassetten unterstützt. Die Software schließt einen MBasic-ähnlichen, also recht komfortablen Basic-Interpreter ein. Ähnlich wie bei MS-DOS ist die Hardware-Struktur eines MSX-Rechners weitgehend vorgegeben, z. B. Kassettenrecorder- und Joystick-Anschluß, TI-Videocontroller usw. Während in Japan MSX-Heimcomputer eine gewisse Verbreitung erlangten, blieb ihr Marktanteil in Europa vergleichsweise gering.

Oasis

Wie Unix ist Oasis ein Mehrbenutzer-Betriebssystem; die Kommando-Syntax ist jedoch benutzerfreundlicher aufgebaut und leichter zu merken. Ursprünglich wurde Oasis für 8-Bit-Rechner entwickelt, ist heute aber auch für 16-Bit-Systeme verfügbar. Im Vergleich zu Unix sowie MP/M bzw. Concurrent-CP/M konnte es sich bisher weniger durchsetzen, was zu einem relativ geringen Anwender-Software-Angebot für Oasis führte. Alle gebräuchlichen Programmiersprachen sind dafür jedoch verfügbar.

OS-9

Das Multitasking-Betriebssystem OS-9 wurde von Microware zunächst nur für den 8-Bit-Prozessor 6809 angeboten, inzwischen gibt es aber unter dem Namen OS-9/68k auch eine 68000-Implementation. OS-9 realisiert trotz seines relativ kleinen Systemkerns (nur etwa 4 KByte beim 6809) bereits viele Unix-Eigenschaften wie Filter-Utilities und Pipelining von Kommando-Parametern, ist im Gegensatz zu Unix aber ganz in Maschinensprache geschrieben und für Echtzeit-Abläufe geeignet. Eine besondere Eigenschaft ist, daß der Systemkern als EPROM z. B. auch in Einplatinen-Computern für Multitasking-Zwecke eingesetzt werden kann.

Pick, Relevation

Eigentlich ist Pick nicht nur ein Multitasking-Betriebssystem, sondern enthält auch eine relationale Datenbank (d. h. eine mit Verweislisten arbeitende; daher auch der Name Relevation, der eine für Mikrocomputer geeignete Version bezeichnet). Da-

bei wird eine eigene, datenbank-orientierte Programmiersprache zur Verfügung gestellt, um kommerzielle Branchen-Software mit geringem Arbeitsaufwand erstellen zu können. Leider gibt es dafür allerdings so gut wie keine Standard-Software. Den Versuch, ein Betriebssystem quasi mit eingebauter Anwender-Software zu einer Einheit zu verschmelzen, um eine besonders leicht bedienbare Benutzerschnittstelle zu erhalten, unternahm später z.B. auch Apple mit den Systemen Lisa bzw. Macintosh und, allerdings auf anderer Ebene, Microsoft mit der MS-DOS-Erweiterung „Windows" und Digital Research bei Concurrent-CP/M und GEM.

UCSD-P-System

Dieses Ein-Benutzer-Betriebssystem wurde bereits im Zusammenhang mit der Sprache Pascal erwähnt. Es enthält einen Interpreter für die Befehle einer virtuellen CPU, die den sogenannten P-Code als Maschinensprache versteht. Das eigentliche Betriebssystem ist weitgehend in diesem P-Code geschrieben und daher zum großen Teil maschinenunabhängig. Eine gewisse Schwäche z.B. gegenüber CP/M oder MS-DOS ist, daß Dateien z.B. auf Disketten immer nur zusammenhängend abgespeichert werden können: Das System meldet „Diskette voll", wenn zwar noch genug Platz da wäre, die Datei aber nicht zusammenhängend Platz findet. Wegen der P-Code-Interpretation ist es auch nicht allzu schnell. Auf der anderen Seite ist der menügesteuerte Aufbau sehr benutzerfreundlich.

Versa-DOS

Wie OS-9 ist das für den Prozessor 68000 geschriebene Versa-DOS von Motorola ein Echtzeit-Multitasking-Betriebssystem, jedoch wesentlich umfangreicher und komfortabler. Wie bei Unix ist das Vorhandensein einer Festplatte praktisch Voraussetzung für den Betrieb von Versa-DOS. Da es nicht mit dem Hintergedanken entwickelt wurde, es später auf ganz andere CPU-Typen zu übertragen, enthält es auch eine Reihe maschinennaher Kommandos wie z.B.:

GTSEG	Neuen Code in den Adressenbereich einer Task bringen
DESEG	Adressenbereich einer Task löschen
DCLSHR	Gemeinsamen Bereich für mehrere Tasks zuweisen
CISR	Interrupt-Routine aktivieren
SINT	Simulation eines Interrupts durch eine Task
RTE	Rückkehr aus einem Interrupt (= einer Exception)

Wie iRMX ist Versa-DOS vorwiegend im industriellen Bereich, z.B. bei Entwicklungssystemen, zu finden und spielt im Bereich kommerzieller Anwendungen so gut wie keine Rolle.

4.2 Herstellerspezifische Betriebssysteme

4.2.1 Überblick

Noch bevor es hersteller-unabhängige Mikrocomputer-Betriebssysteme gab, hatten die Hardware-Hersteller sich natürlich selbst schon Gedanken über passende Disketten-Treiberprogramme gemacht. Zum einen wurden Systeme, die zunächst nur für den Kassettenbetrieb gedacht waren (z.B. Apple-II, Commodore-PET), mit Disketten-Laufwerken nachgerüstet, zum anderen kamen Computer mit Prozessoren auf den Markt, für die es einfach noch kein Standard-Betriebssystem gab.

Insgesamt zeichnet sich heute allerdings immer mehr ein Trend zu Standard-Betriebssystemen ab, wie sie im letzten Kapitel vorgestellt wurden. Nur dann hat man nämlich zumindest eine Chance, durch kompatible Wahl der physikalischen Disk-Aufzeichnungs-Parameter einen Software- und Datenaustausch per Datenträger zwischen unterschiedlichen Computern vorzunehmen und auch auf eine breite Palette von Anwender-Software zurückgreifen zu können, die von Fremdfirmen erstellt wurde. Wo nicht von Haus aus ein Standard-Betriebssystem gewählt wurde, ist oft zumindest eines alternativ zum hersteller-eigenen einsetzbar (z.B. Olivetti M-20: PCOS oder MS-DOS; Apple-II: DOS 3.3 oder, mit Z80-Karte, CP/M-80). Nur in wenigen Fällen werden neu entwickelte Computer noch mit zum Rest der Welt inkompatiblen Betriebssystemen eingeführt (der Apple-Macintosh ist ein bekanntes Beispiel).

Zunächst aber, in Fortsetzung zu der Liste aus Kapitel 4.1.4, eine Auswahl hersteller-gebundener Betriebssysteme.

Amiga-DOS

Betriebssystem des auf der CPU 68000 basierenden Computers „Amiga" von Commodore. Es erlaubt wie MS-DOS und Xenix hierarchische Unter-Directories und unterstützt auch die grafischen Fähigkeiten des Rechners.

Apple-DOS 3.3

Nach wie vor am meisten verbreitetes Betriebssystem für die Computer der Apple-II-Klasse (es gibt hierbei auch Nachbauten z.B. aus Taiwan, so daß man fast von einem hersteller-unabhängigen Betriebssystem sprechen könnte!). Die Aufzeichnung erfolgt mit 35 Spuren zu je 16 Sektoren, was 140 KByte entspricht, und schneller als bei DOS 3.2. Durch Ändern einiger Bytes in DOS 3.3 kann man auch, wenn es das Laufwerk zuläßt, auf mehr Spuren zugreifen (z.B. doppelseitiges Laufwerk mit 2×80 Spuren ergibt 560 KByte).

Commodore-DOS

Bei den 6502-Rechnern CBM, VC-20 und C-64 von Commodore wurde ein ungewöhnlicher Weg gegangen: Die nötigen Disketten-Treiberprogramme sind im wesentlichen nicht im Computer selbst, sondern in dem mit einem eigenen Mikroprozessor ausgestatteten Disk-Laufwerk untergebracht. Das Laufwerk ist mit dem Computer über einen parallelen oder (nicht dem Standard entsprechenden) seriellen IEC-Bus verbunden. Die erreichbare Transfergeschwindigkeit ist dadurch wesentlich geringer als bei Systemen, bei denen der Floppy-Controller direkt mit Daten- und Adressenbus des Computers verbunden ist. Trotzdem hat das Konzept einige Vorteile, z.B. die leichte Erweiterbarkeit mit zusätzlichen Laufwerken, das Ansprechen aller Peripheriegeräte (also auch der Floppy) mit einheitlichen Befehlen und die Möglichkeit, Floppy-Operationen (z.B. Formatieren) den intelligenten Controller selbst durchführen zu lassen, während der Computer schon weiterrechnet.

PCOS

Betriebssystem des Olivetti-Tischcomputers M-20, der mit dem Zilog-16-Bit-Prozessor Z8000 arbeitet. Obwohl für PCOS eine Reihe guter kommerzieller Anwender-Programme existiert, hat sich das System nicht behaupten können; 1984 wendete Olivetti sich daher MS-DOS zu, indem der Rechner eine zusätzliche 8086-Karte erhielt.

Prodos

Von Apple für die Rechner Apple-IIe und IIc (CPU: 6502) entwickeltes Betriebssystem, das jedoch wesentlich mehr Arbeitsspeicher als DOS 3.3 „frißt". Deshalb wird ein Teil von Prodos in das „hinter" dem Basic-ROM liegende RAM geladen, worauf dann durch Bankumschaltung zugegriffen wird. Prodos ist aufzeichnungskompatibel mit DOS 3.3 und besitzt einige zusätzliche Kommandos sowie (wie MS-DOS) Directory-Bäume.

Prologue

Dieses in Frankreich entwickelte Betriebssystem wird von Bull und Olympia für die Tischcomputer Boss und People (CPU: 8086) eingesetzt. Es ist praktisch Bestandteil eines an der kommerziellen Datenverarbeitung orientierten Basic-Interpreters.

SOS

Von Apple für den recht erfolglos gebliebenen Apple-III (CPU: 6502) entwickeltes Betriebssystem, das gegenüber DOS 3.3 eine wesentlich verbesserte Massenspeicher-Verwaltung (auch für Festplatten) besitzt. Das später für den Apple-IIe eingeführte Prodos ist mit SOS kompatibel, so daß ein Disketten-Austausch zwischen Apple-III und Apple-IIe möglich ist.

TRSDOS

Von Tandy (Radio Shack, USA) entwickeltes Betriebssystem für den Tischcomputer TRS-80/I, das auch bei dem dazu kompatiblen Computer Genie-II von Eaca (Hongkong) Verwendung fand. TRSDOS spielt heute aber wie PCOS kaum noch eine Rolle, da Tandy selbst inzwischen Standard-Betriebssysteme bevorzugt.

TOS

So nannte Atari sein Betriebssystem für die auf der CPU 68000 basierende ST-Serie — nach dem 1985 zu Atari gekommenen Jack Tramiel (Tramiel's Operating System). Das TOS unterstützt ähnlich wie MS-DOS hierarchische Sub-Directories und ist selbst in den Nummern der Funktionsaufrufe sowie in der Disketten-Organisation sehr eng an MS-DOS angelehnt, aber eben für einen ganz anderen Mikroprozessor geschrieben.

4.2.2 Apple-DOS 3.3

Wie z.B. CP/M oder MS-DOS besteht auch Apple-DOS (Disk Operating System) aus internen Kommandos, deren zugehörige Routinen schon nach dem „Booten" (Laden des Betriebssystems) resident im Arbeitsspeicher stehen, und zusätzlichen Hilfsprogrammen, die erst bei Bedarf von Diskette geladen werden müssen. Das geschieht allerdings nicht einfach durch Eintippen des jeweiligen Namens, man muß vielmehr RUN bzw. BRUN vorausstellen.

Das „Booten"

Die für den Anschluß eines Disk-Laufwerks erforderliche Controller-Karte darf in einem beliebigen Slot zwischen 1 und 7 sitzen; normalerweise steckt sie jedoch in Slot 6. Eine solche Karte erlaubt den Anschluß von zwei Laufwerken. Beim Einschalten des Computers sucht das Monitorprogramm zunächst von 7 abwärts alle Slots ab, ob ein Disk-Controller vorhanden ist. Wenn nicht, wird lediglich ein Sprung zum Basic-Interpreter durchgeführt, andernfalls wird DOS von Laufwerk 1 geladen. Mit dem Basic-Befehl PR# n oder IN# n kann man auch jederzeit den Boot-Vorgang von Slot n erzwingen, vom Monitorprogramm aus mit n CTRL-P oder n CTRL-K. Nach dem Booten steht für den Basic-Interpreter natürlich am oberen Speicherende weniger Platz zur Verfügung; damit DOS nicht von Basic-Variablen überschrieben werden kann, setzt es automatisch den HIMEM-Pointer (hex 73, 74) auf die DOS-Anfangsadresse statt auf das RAM-Ende.

Eine Besonderheit von Apple-DOS 3.3 ist, daß nach dem Booten automatisch ein Basic-Programm nachgeladen und gestartet wird, dessen Namen man beim Formatieren einer neuen Diskette (wobei auch das DOS auf die Systemspuren 0...2 geschrieben wird) definieren muß. Soll er z.B. HELLO lauten, so formatiert man die neue Diskette mit dem Befehl INIT HELLO. Übrigens müssen beim Apple-II

alle Befehle, auch DOS-Kommandos, grundsätzlich in Großbuchstaben geschrieben sein, sonst werden sie nicht korrekt decodiert.

Belegte Adressenbereiche

Wenn die sogenannte Master-Disk (wie sie von Apple geliefert wird) gebootet wird, so startet ihr HELLO-Programm seinerseits ein Maschinenprogramm, das erst nachsieht, wieviel Arbeitsspeicher im System zur Verfügung steht, und dann das Betriebssystem an das obere Speicherende verschiebt. Stellt man sich später mit dem INIT-Kommando selbst sogenannte Slave-Disketten her, so wird deren Betriebssystem immer gleich an die richtige Stelle im Speicher geladen.

Bei einem System mit 48 KByte Speicher oder mehr sieht die Adressenbelegung durch DOS 3.3 dann hexadezimal wie folgt aus:

03D0—03D2	Sprung zur DOS-Warmstart-Routine
03D3—03D6	Sprung zur DOS-Kaltstart-Routine
03D6—03D8	Sprung zum File-Manager (Dateiverwaltungs-Routinen)
03D9—03DB	Sprung zur RWTS-Routine (Read Track/Sector)
03DC—03F4	wird durch DOS belegt
9600—9CF8	3 DOS-Puffer mit je 595 Bytes (s. MAXFILES)
9CF9—9CFF	nicht verwendet (7 Byte)
9D00—BFFF	Betriebssystem-Routinen

Beim Booten „klinkt" sich DOS in die Ein- und Ausgabe-Vektoren des Monitorprogramms ein (Zellen hex 36...39). Dadurch kann das Betriebssystem Tastatur-Eingaben und PRINT-Ausgaben „mithören", um darin vorkommende Kommandos zu decodieren. Stellt man fest, daß plötzlich keine DOS-Kommandos mehr angenommen werden, so kann man das erneute „Einklinken" erzwingen, indem man mit CALL-151 in den Monitor geht und dann 3D0 G eingibt (DOS-Warmstart).

Aus dem gleichen Grund ist es auch unter DOS nicht mehr zulässig, innerhalb eines Programms einen Befehl wie PR# 1 anzuwenden, da der Basic-Interpreter dann den Ausgabevektor gewaltsam von DOS abhängen würde (bei direkter Eingabe über die Tastatur tritt dieses Problem nicht auf). Statt dessen ist innerhalb des Programms z.B. zum Einschalten eines Drucker-Interface in Slot 1 zu schreiben:

```
260 PRINT CHR$(4);"PR# 1"
```

Und ausgeschaltet wird der Drucker später mit dem Befehl:

```
350 PRINT CHR$(4);"PR# 0"
```

Wenn derartige Befehle häufig im Programm vorkommen, ist es einfacher, am Programmanfang eine Variable wie z.B. D$ zu definieren, die den Zeichencode 4 enthält (dazu gleich mehr):

```
10   D$=CHR$(4) ...
260  PRINT D$"PR# 1"
350  PRINT D$"PR# 0"
```

Kommando-Eingabe und Fehlermeldungen

Von der Tastatur gibt man die DOS-Kommandos wie CATALOG, LOAD, SAVE usw. einfach so ein, wie sie geschrieben werden. Will man sie dagegen von einem Programm aus verwenden, so ist auch das im Vergleich z.B. zu CP/M äußerst einfach. Man braucht nämlich der Zeichenfolge lediglich ein ASCII-Steuerzeichen mit dem Code 04 vorauszustellen. Das letzte vorher ausgegebene Zeichen muß allerdings immer ein Return-Zeichen sein. Folgende Anweisung innerhalb eines Basic-Programme gibt das Disk-Inhaltsverzeichnis aus:

```
30  PRINT "DISK-INHALTSVERZEICHNIS:"
40  PRINT CHR$(4);"CATALOG"
```

Da die PRINT-Anweisung in Zeile 30 mit einem Return-Zeichen endet, ist die korrekte Ausführung von Zeile 40 sichergestellt. Würde dagegen am Ende von Zeile 30 ein Strichpunkt stehen, so würde der DOS-Befehl in Zeile 40 nicht erkannt, ebenso, wenn in einer Zeile 35 ein GET-Befehl zum Einlesen einzelner Zeichen von der Tastatur stehen würde. Die Methode des Aufrufs von DOS-Befehlen durch Voranstellen des Zeichencodes 4 (entspricht CTRL-D) ist nicht nur in Basic-, sondern auch in Maschinenprogrammen anwendbar. Im Gegensatz zu Basic-Befehlen dürfen nicht mehrere DOS-Kommandos nacheinander durch Doppelpunkte getrennt in einer Zeile stehen.

Dateinamen müssen mit einem Zeichen beginnen, dessen Code mindestens hex 40 ist (Klammeraffe bzw. Paragraph-Zeichen oder beliebiger Buchstabe). Die dann folgenden Zeichen des bis zu 30 Zeichen langen Namens dürfen auch Ziffern, Satz- oder Steuerzeichen sein. Dadurch kann man sogar „unsichtbare" Dateinamen erzeugen:

```
PRINT CHR$(4)+"SAVE A"+CHR$(8)+CHR$(32)
```

Dieser Befehl speichert ein Basic-Programm unter einem im Directory unsichtbaren Namen ab, der aus dem Buchstaben A, einem Backspace-Steuerzeichen (Cursor eine Stelle zurück) und einem Leerraum besteht, so daß das A auf dem Bildschirm (jedoch nur dort) vom Leerraum überschrieben (gelöscht) wird.

Stellt DOS einen Fehler fest, so wird dieser in englischem Klartext ausgegeben und ein laufendes Programm beendet. Mit ON ERR GOTO kann man ihn allerdings in Basic auch abfangen; den Fehlercode kann man dann mit PEEK(222) holen:

Code	Fehlermeldung	Bedeutung
0	—	kein Fehler
1	Language not available	Gewünschte Sprache nicht im Speicher
2, 3	Range error	Unzulässige Parameter
4	Write protected	Disk ist schreibgeschützt
5	End of data	INPUT oder GET nach Dateiende
6	File not found	Datei nicht gefunden
7	Volume mismatch	Volume-Nr. der Disk paßt nicht
8	I/O error	Diskette nicht ansprechbar
9	Disk full	Kein Platz auf der Diskette mehr
10	File locked	Datei mit LOCK schreibgeschützt
11	Syntax error	Befehl falsch geschrieben
12	No buffers available	MAXFILES ist zu gering gesetzt
13	Filetype mismatch	Dateityp paßt nicht
14	Program too large	HIMEM zu niedrig gesetzt
15	Not direct command	Befehl nur innerhalb eines Programms zulässig (APPEND, OPEN, POSITION, READ, WRITE)

Hinter einem Dateinamen kann man, jeweils durch Kommata getrennt, auch eine neue Slot- und Laufwerksnummer (Drive) angeben. Fehlen diese Angaben, wird jeweils das beim Booten bzw. zuletzt verwendete Laufwerk angesprochen:

LOAD PGM	Lädt das Basic-Programm PGM vom zuletzt angesprochenen Laufwerk
LOAD PGM,D2	Lädt PGM vom Laufwerk 2 des gleichen Slots
LOAD PGM,S5,D2	Lädt PGM vom Laufwerk 2 in Slot 5

DOS-Kommandos

Im folgenden werden die Kommandos von DOS 3.3 aufgeführt. Für Dateinamen wird „name" verwendet (die Eingabe erfolgt stets ohne Anführungszeichen!), statt einer expliziten Zeichenfolge kann aber auch eine Basic-Variable als Dateiname dienen. Optionale Angaben stehen jeweils in spitzen Klammern. „v" ist eine Zahl 1...254, die die Volume-Nummer einer Diskette angibt: DOS verweigert den Zugriff, wenn nach einem Befehl eine andere Volume-Nummer als die der aktuellen Diskette angegeben wird. „s" kennzeichnet eine Slot-Nummer (1...7, Slot 0 ist für eine RAM-Karte reserviert), „d" die Nummer eines Disk-Laufwerks (1 oder 2). Andere Angaben werden beim jeweiligen Kommando erläutert.

Auf einen besonderen Effekt der Kommandos BSAVE, SAVE, OPEN und CLOSE muß vorher noch hingewiesen werden: Wird eine Datei erzeugt, mit deren Namen schon eine auf der Diskette existiert, so wird die alte keineswegs wirklich gelöscht. Vielmehr wird sie nur überschrieben. Wenn nun die neu erzeugte kürzer

ist als die alte, so werden im Inhaltsverzeichnis noch alle Sektoren der alten Datei als belegt gekennzeichnet (!). Bei Textdateien kann dann sogar der Fall auftreten, daß man beim Lesen der neuen (kürzeren) Datei über deren Ende hinaus noch Reste der alten einliest. Es ist daher unbedingt empfehlenswert, eine alte Datei mit DELETE zu löschen, bevor sie mit dem gleichen Namen überschrieben bzw. neu eröffnet wird.

APPEND name <,Ss> <,Dd> <,Vv>
Wirkt wie OPEN, setzt jedoch den Zeiger nicht an den Anfang, sondern an das Ende der angegebenen Datei, um sie zu verlängern. Nach APPEND name muß unbedingt WRITE name folgen.

BLOAD name <,Aa> <,Ss> <,Dd> <,Vv>
Laden eines Maschinenprogramms von Diskette. Wenn es nicht an die gleiche Adresse geladen werden soll, ab der es abgespeichert wurde, kann man die gewünschte Anfangsadresse nach A dezimal oder, durch ein Dollar-Zeichen gekennzeichnet, hexadezimal angeben, z.B.:
BLOAD PRG1,A$900,D2

BRUN name <,Aa> <,Ss> <,Dd> <,Vv>
Wie bei BLOAD wird zunächst ein Maschinenprogramm von Disk geladen, dann aber sofort bei seiner Anfangsadresse gestartet. Vorsicht: Wenn das Programm durch Nennen einer abweichenden Anfangsadresse verschoben geladen wird, ist es mit größter Wahrscheinlichkeit nicht mehr ausführbar, so daß der Rechner sich „aufhängt".

BSAVE name,Aa,Ln <,Ss> <,Dd> <,Vv>
Abspeichern des Bereichs mit der Anfangsadresse a und der Länge n Byte. Wie übrigens auch bei allen anderen DOS-Befehlen können die Parameter durch ein vorangestelltes Dollar-Zeichen auch hexadezimal eingegeben werden, z.B.:
BSAVE PRG1,A$300,L$BF

CATALOG <,Ss> <,Dd>
Gibt das Disk-Inhaltsverzeichnis aus. Vor den Dateinamen steht der Dateityp sowie die Anzahl von Sektoren (je 256 Byte), die eine Datei belegt. Die unter DOS 3.3 definierten Dateitypen sind:

A Applesoft-Basic-Programm (LOAD, RUN)
B Maschinenprogramm (siehe BLOAD, BRUN)
I Integer-Basic-Programm (LOAD, RUN)
T Textdatei (OPEN, READ, WRITE, CLOSE, POSITION, EXEC)

Als Überschrift wird bei DOS 3.3 die bei INIT festgelegte Volume-Nummer angezeigt, bei den DOS-kompatiblen Betriebssystemen MC-DOS II und ZDOS dagegen die Anzahl der noch nicht belegten Sektoren der Disk.

CHAIN name <,Ss> <,Dd> <,Vv>
Ermöglicht das Laden und Starten eines neuen Integer-Basic-Programms, ohne die Variablen-Inhalte des vorherigen zu löschen. Bei Applesoft-Basic ist CHAIN leider nicht anwendbar.

CLOSE <name>
Die Datei „name" wird nach OPEN geschlossen. Falls kein Name angegeben ist, werden alle offenen Dateien geschlossen, jedoch keine EXEC-Dateien, die gerade gelesen werden.

DELETE name <,Ss> <,Dd> <,Vv>
Löscht die Datei name, falls sie nicht mit dem Kommando LOCK vor Änderungen geschützt wurde; ggf. ist vorher UNLOCK name einzugeben. Falls die Datei dabei noch geöffnet war, wird auch ein CLOSE simuliert. Der Versuch, eine nicht existente Datei zu löschen, führt zu einer Fehlermeldung.

EXEC name <,Rr> <,Ss> <,Dd> <,Vv>
Schaltet die Tastatur-Eingabe auf die Datei „name" um, optional beginnend mit der r-ten Zeile dieser Datei. Damit kann man z.B. Demo-Programme laufen lassen, indem man Tastatur-Eingaben simuliert. Aber Vorsicht: Auch mit Reset ist ein Abbruch dieses Vorgangs nicht mehr möglich, solange die Datei nicht zu Ende ist; es hilft nur noch ein Ausschalten des Rechners! Eine weitere nützliche Anwendung ist, ein z.B. per Telefon-Modem empfangenes und auf Diskette als Text-File (Typ T) abgespeichertes Basic-Programm in ein lauffähiges Programm umzuwandeln; man braucht dazu nur im Basic-Interpreter EXEC name einzugeben.

FP <,Ss> <,Dd> <,Vv>
Schaltet, falls man sich in Integer-Basic befand (das bei 64-KByte-Systemen ins RAM ab D000 geladen werden kann), auf Applesoft-Basic um, löscht ein eventuell gespeichertes Basic-Programm und initialisiert alle Zeiger neu (vgl. INT). Wenn das ROM Integer-Basic enthält, versucht DOS, den Applesoft-Interpreter unter dem Namen APPLESOFT von Diskette zu laden (er befindet sich auf der Master-Systemdiskette).

HIDE name <,Ss><,Dd> oder MODE <,Ss><,Dd>
Nicht bei Apple-DOS 3.3, sondern nur bei ZDOS und MC-DOS II vorhanden. Mit HIDE (MC-DOS) bzw. MODE (ZDOS) kann man eine Datei im Inhaltsverzeichnis (CATALOG) unsichtbar und bei Bedarf (wenn man dann noch ihren Namen kennt!) wieder sichtbar machen. Falls einem der Name nicht mehr einfällt: Mit dem Hilfsprogramm FID (siehe später) kann man durch Verwenden von „=" statt des Dateinamens auch solche Dateien bearbeiten.

IN# s
Lenkt den Eingabe-Vektor des Apple-II von der Tastatur auf die Karte in Slot s um, ohne (im Gegensatz zum gleichlautenden Basic-Befehl) die DOS-I/O-Routinen „abzuhängen".

INIT name <,Vv> <,Ss> <,Dd>
Formatiert eine neue oder gebrauchte Diskette. Dabei werden vorher vorhandene Daten gelöscht. Ferner wird auf die Systemspuren 0...2 das Betriebssystem geschrieben. „name" gibt an, wie das Basic-Programm heißt, das nach dem Booten dieser Disk automatisch gestartet werden soll. Wird die Volume-Nummer v weggelassen, so wird dafür 254 angenommen. Dieser Befehl existiert bei ZDOS und MC-DOS II nicht; statt dessen gibt es MODE bzw. HIDE (s. d.).

INT
Schaltet, falls man sich in Applesoft-Basic befand, auf Integer-Basic um (das bei 64-KByte-Systemen ins RAM ab D000 geladen werden kann), löscht ein eventuell gespeichertes Basic-Programm und initialisiert alle Zeiger neu (vgl. FP).

LOAD name <,Ss> <,Dd> <,Vv>
Lädt ein Basic-Programm von Diskette ins RAM, ohne es jedoch zu starten.

LOCK name <,Ss> <,Dd> <,Vv>
Schützt eine Datei vor versehentlichem Überschreiben, Umbenennen oder Löschen.

MAXFILES n
Definiert die Zahl n (1...16) der Datenpuffer, die DOS unterhalb von hex 9B00 (bei 48- oder 64-KByte-Systemen) zur Verfügung stehen, und somit auch die Zahl der Dateien, die gleichzeitig geöffnet sein dürfen. Je nach MAXFILES wird von DOS auch der HIMEM-Zeiger des Basic-Interpreters entsprechend gesetzt. Vorsicht: Da auch z.B. CATALOG einen Puffer benötigt, wird eine Fehlermeldung ausgegeben, wenn MAXFILES 1 eingegeben wurde, eine Datei bereits offen ist und nun das Inhaltsverzeichnis angezeigt werden soll.

MON <I> <O> <C>
Erlaubt das Überwachen (monitoring) aller Ein- (I) und Ausgaben (O) sowie Steuerbefehle (C) von DOS. Gibt man mehrere Parameter hinter MON an, sind sie durch Kommata zu trennen. Beispiele:

MONI	Anzeige aller von DOS gelesenen Daten
MONO	Anzeige der zu DOS transferierten Daten
MON C,I,O	Anzeige aller Ein-/Ausgaben sowie Steuerbefehle

NOMON <I> <O> <C>
Hebt MON (siehe dort) wieder auf. NOMON ohne I, O oder C wird von DOS ignoriert. Nach dem Booten von DOS ist das System im Zustand NOMON I,O,C.

OPEN name <,Lr> <,Ss> <,Dd> <,Vv>
Öffnet eine Datei zum Schreiben oder Lesen. Dabei wird ein Puffer von 595 Byte reserviert und leider (unnötigerweise) auch dann ein Eintrag im Disk-Inhaltsverzeichnis erzeugt, wenn die Datei bisher noch nicht existierte, jedoch versehentlich gelesen werden sollte. Der Befehl OPEN kann jedoch vor READ entfallen, wenn sichergestellt ist, daß der Puffer nicht z.B. von CATALOG oder von einer weiteren

Datei in Anspruch genommen wird. Bei Random-Access-Dateien (Dateien mit wahlfreiem Zugriff) wird mit dem Parameter r nach dem Buchstaben L die Länge eines Records festgelegt.

POSITION name <,Rr>
Setzt den Datei-Zeiger auf den Anfang der r-ten Zeile ab der aktuellen Zeile (also eine relativ auf die momentane Zeile bezogene Angabe!). Dieser Befehl kann nach OPEN und vor READ bzw. WRITE stehen.

PR# s
Lenkt den Ausgabe-Vektor des Apple-II vom Bildschirm auf die Karte in Slot s um, ohne (im Gegensatz zum Basic-Befehl PR# s) die DOS-Routinen „abzuhängen".

READ name <,Rr> <,Bb>
Dieser Befehl leitet die Basic-Befehle INPUT und GET von der Tastatur auf die Textdatei „name" um, optional auch ab dem b-ten Zeichen dieser Datei (relative Anfangsposition). Wie WRITE wird READ von jedem anderen DOS-Befehl oder mit PRINT CHR$(4) aufgehoben. Bei Random-Access-Dateien legt der Parameter r fest, ab welchem Record der Datei gelesen werden soll.

RENAME name1,name2 <,Ss><,Dd><,Vv>
Die Datei name1 wird (unabhängig von ihrem Typ) in name2 umbenannt. Vorsicht: Wenn name2 schon existiert, gibt es im Inhaltsverzeichnis plötzlich zwei Dateien mit diesem Namen, von denen nur noch die erste geladen oder gelöscht werden kann. Ferner darf man dem Basic-Programm, das beim Booten geladen wird (z.B. HELLO) keinen anderen Namen geben, da er mit INIT innerhalb des Betriebssystems mit auf die Diskette geschrieben wurde.

RUN name <,Ss> <,Dd> <,Vv>
Wie LOAD, jedoch wird das Basic-Programm sofort nach dem Laden auch automatisch gestartet.

SAVE name <,Ss> <,Dd> <,Vv>
Speichert das im RAM stehende Basic-Programm auf Diskette ab.

UNLOCK name <,Ss> <,Dd> <,Vv>
Hebt den Schreibschutz einer Datei (LOCK) wieder auf.

VERIFY name <,Ss> <,Dd> <,Vv>
Prüft, ob eine Datei lesbar auf der Diskette abgespeichert wurde (die Übereinstimmung mit dem Inhalt des Arbeitsspeichers erfolgt im Gegensatz zu manchen anderen Computern nicht!).

WRITE name <,Rr> <,Bb>
Nach OPEN wird damit dem DOS mitgeteilt, daß die geöffnete Datei geschrieben werden soll. Mit b kann optional eine relative Position innerhalb einer schon existierenden Datei angegeben werden, ab der sie überschrieben bzw. verlängert wer-

den soll. Nach WRITE werden alle Ausgaben nicht mehr auf den Bildschirm, sondern auf die Datei geleitet. Jeder andere DOS-Befehl oder INPUT hebt WRITE wieder auf; es genügt auch eine Anweisung wie PRINT CHR$(4). Bei Random-Access-Dateien legt der Parameter r fest, ab welchem Record der Datei mit dem Schreiben begonnen werden soll.

Hilfsprogramme

Die folgenden Programme werden auf der Apple-DOS-Master-Diskette mitgeliefert und übernehmen Aufgaben, die vom Betriebssystem-Kern nicht wahrgenommen werden können.

COPY und COPYA

Während COPY für Integer-Basic geschrieben wurde, läuft COPYA unter dem (normalerweise verwendeten) Applesoft-Basic. Der Start erfolgt mit RUN. Beide Programme dienen zum Kopieren ganzer Disketten und laden automatisch ein Maschinenprogramm nach (COPY.OBJ). Die Zieldiskette wird automatisch vorher formatiert. Das Kopieren erfolgt rein physikalisch, d.h. die Dateien belegen auf der neuen Diskette genau die gleichen Spuren und Sektoren wie auf der alten.

Das Kopieren von Disketten ist sowohl mit zwei als auch mit einem Laufwerk möglich. Im letzteren Fall wird man regelmäßig aufgefordert, die Disketten zu wechseln, da der Arbeitsspeicher, der als Puffer verwendet wird, kleiner ist als die Kapazität einer Diskette.

Da COPY bzw. COPYA grundsätzlich alle Spuren und Sektoren einer Diskette kopiert, funktioniert das Kopieren mit FID u.U. schneller, wenn nur sehr wenige Dateien vorhanden sind.

FID

So heißt der „File Developer"; er wird mit BRUN FID gestartet und meldet sich dann mit einem Menü. Er erlaubt u.a. das Kopieren, Löschen und Prüfen einzelner Dateien und Datei-Gruppen. Das Kopieren ist nicht nur mit zwei, sondern auch mit einem Laufwerk möglich; man wird dann nach jeder Datei gebeten, die Diskette zu wechseln und dann eine Taste zu drücken.

Bei FID ist es im Gegensatz zu den normalen DOS-Befehlen möglich, innerhalb oder statt eines Dateinamens ein Gleichheitszeichen zu verwenden, das dann ein oder mehrere Zeichen als nicht relevant markiert. Beispiele:

=M=	Alle Namen, die ein M enthalten
MA=	Alle Namen, die mit MA beginnen
=	Alle Dateien der Diskette

Wenn man das Gleichheitszeichen (es entspricht dem CP/M-Wildcard-Sternchen) verwendet, fragt FID vor der Kommandoausführung „Do you want prompting?". Gibt man jetzt Y ein, so wird man bei jeder dann angewählten Datei gefragt, ob auch diese gemeint sei, was wiederum mit Y und N beantwortbar ist. Mit Q kann man den Vorgang dagegen dann ganz abbrechen.

FID ist auch dann nützlich, wenn man eine ganze Diskette kopieren möchte. Wenn man länger mit einer Disk gearbeitet hat, haben sich viele Dateien angesammelt, die nicht zusammenhängend auf ihr stehen, sondern in Teile zerrissen, die immer dort hineingezwängt wurden, wo noch freie Sektoren waren. Wurden zum Beispiel mehrere kleinere Dateien gelöscht, die an ganz unterschiedlichen Stellen der Diskette waren, so wird eine neue, längere Datei dann genau in die entstandenen Lücken abgespeichert. Das Laden dieser neuen Datei dauert dann unnötig lang, weil sich der Laufwerks-Kopf oft hin- und herbewegen muß. Kopiert man eine solche Diskette mit COPY bzw. COPYA, so werden alle Spuren unverändert übernommen, so daß sich daran nichts ändert. Nimmt man dagegen eine neue Diskette, formatiert sie mit INIT und kopiert auf sie dann alle Dateien mit „=" als Name, so werden sie physikalisch zusammenhängend abgespeichert, so daß das Laden der einzelnen Dateien weniger lang dauert. (Diese Überlegung gilt selbstverständlich auch für andere Betriebssysteme.)

MUFFIN

Mit BRUN MUFFIN kann man Dateien von einer DOS-3.2-Diskette im 13-Sektor-Format auf eine DOS-3.3-Disk mit 16 Sektoren kopieren. Das Programm ist wie COPY, COPYA und FID menügesteuert. Ähnlich wie bei FID kann man auch das Wildcard-Zeichen „=" in Dateinamen verwenden. Da heute aber so gut wie keine Software mehr auf DOS-3.2-Disketten angeboten wird, ist MUFFIN eigentlich nicht mehr nötig.

Die RWTS-Routine (Read/Write Track/Sector)

Von einem Maschinenprogramm aus kann man bestimmte Sektoren der Diskette direkt beschreiben. Das kann recht nützlich sein, wenn man beispielsweise Fehler „reparieren" oder ein neues DOS auf die Systemspuren laden möchte. Hierzu gibt es die RWTS-Routine. Da je nach Arbeitsspeicher-Größe das DOS an unterschiedlichen Adressen stehen kann, schreibt es beim Booten automatisch einen Sprung zu dieser Routine in die Adresse 03D9. Bei 03E3 findet sich schließlich ein kurzes Unterprogramm, das die Adresse der zugehörigen Parameter-Tabelle (IOB = Input/Output Control Block) in die CPU-Register A und Y lädt. Diese Tabelle umfaßt 17 Byte, die hier relativ zur Anfangsadresse gezählt werden (B7E8 bei 48-KByte-Systemen):

Byte	Inhalt
0	Aufzeichnungstyp, nur 01 zulässig
1	Slot des Controllers mal 16 (hex 60 für Slot 6)
2	Laufwerksnummer (01 oder 02)
3	Erwartete Volume-Nummer; 00 für beliebige Nummer
4	Spur (hex 00...22)
5	Sektor (hex 00...0F)
6, 7	Startadresse der DCT-Parameter-Tabelle (s. u.)
8, 9	Startadresse des Datenpuffers
10, 11	nicht belegt
12	Befehlscode für RWTS: 00 = Kopf nur positionieren 01 = Sektor in den Datenpuffer einlesen 02 = Datenpuffer auf Sektor schreiben 04 = Formatieren der gesamten (!) Diskette
13	Enthält Fehlercode nach Aufruf von RWTS (hex): 10 = Diskette schreibgeschützt 20 = Volume-Nummer paßt nicht 40 = Physikalischer Fehler 80 = Lesefehler im Daten- oder Adreßfeld (z.B. Datenfeld leer)
14	Von Disk gelesene Volume-Nummer
15	Slot * 16 des zuletzt verwendeten Laufwerks (Slot 6: hex 60)
16	Nummer des zuletzt verwendeten Laufwerks (01 oder 02)

Die im IOB enthaltene Adresse von „DCT" (Device Characteristics Table, Geräte-Spezifikations-Tabelle) ist bei 48-KByte-Systemen B7FB. Diese Parameterliste ist nur vier Byte lang und enthält folgende Angaben:

Byte	Inhalt
0	Gerätecode; normalerweise 00
1	Aufzeichnungs-Code; normalerweise 01
2, 3	Komplement der Motor-Hochlaufzeit in 0,1-ms-Einheiten; normalerweise EF und D8.

Sind alle erforderlichen Parameter gesetzt, braucht man der RWTS-Routine lediglich in den Registern A und Y die Anfangsadresse der IOB-Tabelle zu übergeben und RWTS durch JSR $3D9 aufzurufen. Dabei ist zu bedenken, daß es so ohne weiteres möglich ist, auch Spuren zu lesen und zu ändern, die normalerweise nicht für Daten freigegeben sind:

Spur	Inhalt
0...2	Betriebssystem (auf Spur 2 nur Sektoren 0...4)
3...16	frei für Dateien
17	Sektor 0: Bitmuster mit noch nicht belegten Sektoren (VTOC = Volume Table of Contents); Sektoren 1...15: Disk-Inhaltsverzeichnis

Sektor 0 von Spur 17, also die VTOC, ist folgendermaßen aufgebaut (die Bytes sind hier in hexadezimaler Zählweise angegeben):

Byte	Inhalt
0	nicht verwendet (02)
1, 2	Spur und Sektor des Inhaltsverzeichnis-Anfangs (11 0F)
3	DOS-Versionsnummer (04)
4, 5	nicht verwendet (00 00)
6	Volume-Nummer der Disk (01...FE)
7...26	nicht verwendet (00...00)
27	Max. Zahl der Spur-Sektor-Paare in der TBM (7A; s. u.)
28...2F	nicht verwendet (00...00)
30...33	Maskierungs-Bitmuster für die TBM (FF FF 00 00)
34	Zahl der Spuren einer Diskette (23)
35	Zahl der Sektoren einer Spur (0F)
36, 37	Zahl der Bytes pro Sektor (00 01 = dez. 256)
38...3B	Track Bit Map (TBM, belegte Sektoren) von Spur 00
3C...3F	TBM von Spur 1
40...43	TBM von Spur 2
44...7B	TBM der Spuren hex 03...10 (dez. 3...16)
7C...7F	TBM der Spur hex 11 (dez. 17; VTOC und Inhaltsverzeichnis)
80...C3	TBM der Spuren hex 12...22 (dez. 17...35)
C4...FF	Bei 35-Spur-Disks nicht verwendet

Jedes 1-Bit der Track Bit Map (TBM) markiert einen belegten Sektor. Dadurch werden als Belegungstabelle für eine Spur nur zwei TBM-Bytes benötigt. Das höchstwertige Bit des ersten Bytes entspricht Sektor 0F, das niederwertigste von Byte 2 dem Sektor 00. Als belegt gekennzeichnete Sektoren können von DOS aus nur mit DELETE wieder freigegeben werden, jedoch nicht, wenn man eine Datei unter gleichem Namen mit einem kürzeren File überschreibt (siehe Abschnitt über DOS-Kommandos).

Obwohl Spur 2 nur teilweise von DOS 3.3 belegt ist, wird sie in der VTOC trotzdem vollständig für den Dateizugriff gesperrt. Nach Abzug der Spuren 0, 1, 2 und 17 verbleiben daher an freiem Speicherplatz für Dateien nur noch 124 KByte.

Kommata und Doppelpunkte in Textdateien

Im Applesoft-Basic-Interpreter gibt es keinen Befehl wie LINE INPUT in MBasic, mit dem es möglich ist, auch Zeilen einzulesen, die Kommata oder Doppelpunkte enthalten. Bei Tastatur-Eingaben kann man diesen Befehl jedoch mit Hilfe der Anweisung GET innerhalb eines Unterprogramms simulieren, das in *Abb. 4.2.2.1* bei Zeile 200 beginnt. Zeile 10 enthält einen Test-Aufruf, der eine eingegebene Zeile nochmals auf den Bildschirm ausgibt.

```
10GOSUB200:PRINT"EINGEGEBEN:";S1$:GOTO10
20:
200REM *** LINE INPUT MIT GET ***
210P=0:S1$=""
220GETS2$:IFS2$=CHR$(8)THEN260:REM BACKSPACE
230IFS2$=CHR$(13)THENPRINT:RETURN
240IFS2$<CHR$(32)THEN220:REM KEINE STEUERZ.
250PRINTS2$;:S1$=S1$+S2$:P=P+1:GOTO220
260IFP=0THEN210:REM CURSOR AM ZEILENANF.
270PRINTCHR$(8);" ";CHR$(8);:REM LOESCHEN
280P=P-1:IFP=0THEN210
290S1$=LEFT$(S1$,P):GOTO220
```

Abb. 4.2.2.1
Eingabe einer Zeile, die auch Kommata und Doppelpunkte enthalten darf, beim Apple-II

Möchte man eine solche Zeile auf Diskette speichern und später mit INPUT von dort wieder einlesen, so tritt leider das Problem auf, daß auch hier die Fehlermeldung „Extra ignored" ausgegeben wird, sobald ein Komma oder ein Doppelpunkt gefunden wird, und der Rest der Zeile fehlt. Selbstverständlich könnte man das gerade vorgeschlagene Unterprogramm statt für Tastatur-Eingaben nach OPEN und READ zum Einlesen von Diskette verwenden; allerdings wäre die Geschwindigkeit dabei relativ gering.

Eine Lösung des Problems ist es, schon beim Abspeichern jede Zeile von Anführungszeichen (ASCII dezimal 34, hex 22) einzuschließen. Um z.B. den String S1$, der auch Kommata enthalten darf, mit dem Dateinamen TEST auf Disk zu speichern, eignet sich das Programmbeispiel in *Abb. 4.2.2.2.* Die maximale Zei-

```
10S1$="DAS IST, ALS DEMO, EIN TEST: OK?"
20GOSUB100:REM UNTERPG.ZUM ABSP.VON STRINGS MIT KOMMATA
30END
100D$=CHR$(4):N$="TEST":REM CTRL-D UND DATEINAME
110PRINTD$"OPEN"N$:REM VERMEIDET FEHLERMELDUNG BEI DELETE
120PRINTD$"DELETE"N$:REM EV.ALTE DATEI N$ LOESCHEN
130PRINTD$"OPEN"N$:PRINTD$"WRITE"N$
140PRINTCHR$(34);S1$;CHR$(34):REM S1$ IN ANFUEHRUNGSZ.
150PRINTD$"CLOSE"N$:RETURN
```

Abb. 4.2.2.2 Das Abspeichern und Wiedereinlesen von Strings auf und von Disk, die Kommata enthalten, ist kein Problem, wenn sie von Anführungszeichen (Dezimalcode 34) eingeschlossen werden

```
10GOSUB160:REM EINLESEN EINER ZEILE VON DISK
20PRINTE$:REM ZEILE AUF BILDSCHIRM ANZEIGEN
30END
160D$=CHR$(4):N$="TEST":REM CTRL-D UND DATEINAME
170PRINTD$"READ"N$:REM BEI NUR 1 DATEI KEIN OPEN NOETIG
180INPUTE$:PRINTD$"CLOSE"N$:RETURN

>RUN
DAS IST, ALS DEMO, EIN TEST: OK?
```

Abb. 4.2.2.3 Routine zum Einlesen eines Strings von Disk. Ein OPEN-Befehl ist nicht nötig, wenn nur eine einzige Datei geöffnet wird

chenzahl in einer Zeile darf 255 betragen. Der String wird in *Abb. 4.2.2.3* wieder von der Diskette gelesen und auf den Bildschirm ausgegeben, wobei der INPUT-Befehl die Anführungszeichen am Anfang und am Ende der Zeile automatisch entfernt.

Viele Textverarbeitungs-Programme (z.B. Applewriter) sowie Assembler-Editor-Programme (z.B. Lisa, Big Mac u.a.) vermeiden dieses Problem allerdings, indem Textdateien wie Maschinenprogramme als Speicherbereich (Dateityp B) auf Disk gespeichert werden, was normalerweise auch schneller funktioniert.

4.2.3 Commodore-DOS

In Kapitel 4.2.1 wurde bereits erwähnt, daß Commodore bei seinen 65XX-Computern einen gänzlich anderen Weg als die meisten Rechnerhersteller ging. Die Floppy-Disk-Einheit (CBM-2030/3040/8050, VC-1541) ist ein selbst intelligentes Peripheriegerät mit eigenem Prozessor und Speicher, das mit dem Tischcomputer (CBM, C-64, VC-20, C-16 usw.) nicht über den Adressen- und Datenbus, sondern über einen IEC-Bus-Anschluß oder, beim Laufwerk VC-1541 für die Rechner VC-20, C-64 und C-16, mit einer seriell arbeitenden IEC-Bus-Variante verbunden ist.

Dieses Konzept bedingt einerseits zwar eine geringere Datentransferrate als bei anderen Rechnern, andererseits aber kann die „intelligente" Floppy-Einheit z.B. eine Diskette formatieren, während der Tischcomputer gerade ein Programm bearbeitet.

Disk-Organisation

Während bei den meisten anderen Floppy-Einheiten auf jeder Spur gleich viele Sektoren untergebracht sind (beim Apple-DOS 3.3 zum Beispiel 16), nützte Commodore die Tatsache aus, daß die äußeren Spuren länger sind als die inneren. Je nach Laufwerkstyp ergeben sich für eine Spur die auf der nächsten Seite genannten Sektorenzahlen (Spur 0 ist jeweils die äußerste).

Die Belegungstabelle (Block Availability Map, BAM) entspricht in ihrer Funktion der VTOC bei Apple-DOS 3.3; jedes 1-Bit kennzeichnet einen belegten Sektor. Da es sich bei den Typen CBM-3040/4040 und CBM-8050 um Doppellaufwerke handelt, können die angegebenen Kapazitätswerte mit 2 multipliziert werden.

	CBM-3040/4040, VC-1541:		CBM-8050:	
Sektoren pro Spur	Spur 0—16 17—23 24—29 30—34	Sektoren 21 20 18 17	Spur 0—38 39—52 54—65 66—76	Sektoren 29 27 25 23
Kapazität:	172,5 KByte		520,75 KByte	
Sektoren insgesamt:	690		2083	
Byte pro Sektor:	256		256	
Inhalts-verz.:	Spur 17, 18 Sektoren		Spur 39, 27 Sektoren	
Belegungs-tabelle	Spur 17, 2 Sektoren		Spur 38, 2 Sektoren	

Kommando-Übergabe

Wie bei CP/M können einzelne, nicht relevante Zeichen von Dateinamen mit Fragezeichen, ganze Zeichenfolgen dagegen mit einem Sternchen ersetzt werden. Die Namen dürfen maximal 16 Zeichen lang sein und auch Steuerzeichen enthalten. will man die erste Datei des Inhaltsverzeichnisses als Basic-Programm laden, so kann man z. B. eingeben:

```
LOAD "*",8
```

Dabei ist 8 die IEC-Bus-Adresse des Floppy-Laufwerks.

Ein Teil der Floppy-Disk-Befehle, nämlich jener, der unmittelbar in den Basic-Interpreter eingebunden ist, wurde bereits im Kapitel über Basic aufgeführt (LOAD, SAVE, DOPEN, DIR usw.). In Commodore-Basic 3.X (z.B. C-64) dient der LOAD-Befehl u.a. auch dazu, das Inhaltsverzeichnis einer Diskette anzuzeigen, indem es in den Speicher geladen und dann mit LIST sichtbar gemacht wird:

```
LOAD "$",8
LIST
```

Will man ein anderes Laufwerk als das voreingestellte ansprechen, so braucht man nur dessen Nummer (0 oder 1) dem Dollar-Zeichen nachzustellen. Man beachte, daß das Inhaltsverzeichnis das vorher im Speicher stehende Basic-Programm überschreibt! Der DIR-Befehl von Basic 4.X schaffte endlich diesen Anachronismus ab.

Die anderen Floppy-Kommandos müssen durch direktes Ansprechen des IEC-Bus eingegeben werden. Dazu muß zunächst ein IEC-Bus-Kanal geöffnet werden. Ist dessen Sekundäradresse (d.h. die Zahl, die nach der IEC-Bus-Primäradresse

genannt wird), gleich 15, so weiß das Laufwerk, daß nun Steuerkommandos folgen, und ein Dateiname muß bei OPEN nicht angegeben werden.

Ist die Sekundäradresse dagegen nicht 15 (beliebiger anderer Wert), so muß hinter ihr noch ein Dateiname stehen, denn damit wird eine Disk-Datei zum Schreiben oder Lesen geöffnet. (In Commodore-Basic 4.X stehen dafür durchaus komfortablere Kommandos wie DOPEN zur Verfügung, der „direkte" Weg über den IEC-Bus ist jedoch ebenso noch möglich.)

Die folgenden Anweisungen formatieren eine fabrikneue Diskette in Basic 3.X neu (in Basic 4.X gibt es dafür den Befehl HEADER):

```
OPEN 1,8,15
PRINT#1,"NEW0:SPIELPROGRAMME,01"
CLOSE 1
```

Hierbei wird zunächst die logische Datei 1 auf der IEC-Primäradresse 8, mit der die Disk-Einheit angesprochen wird, der Kommando-Kanal (Sekundäradresse 15) geöffnet. Dann transferiert der Befehl PRINT# 1 das von Anführungszeichen eingeschlossene Kommando zum Laufwerk. NEW bedeutet Formatieren (läßt sich mit N abkürzen), die folgende Ziffer 0 bedeutet Laufwerk 0. Nach dem Doppelpunkt folgt der Name der Disk, der in jedem Inhaltsverzeichnis invers als Überschrift dargestellt wird und maximal 16 Zeichen umfassen darf, sowie die Archivierungs-Nummer der Diskette, hier willkürlich 01. Das Laufwerk 0 läuft an und formatiert die Diskette in etwa zwei Minuten. Auch vor Beendigung dieses Vorgangs darf man schon CLOSE 1 (oder nur CLOSE) eingeben, da das Formatieren intern im Floppy-Laufwerk abläuft und der Computer selbst dazu nichts mehr beiträgt.

Fehlermeldungen

Trat bei einem Disk-Kommando ein Fehler auf, so fängt die Leuchtdiode am Laufwerk zu blinken an, zunächst wird aber am Rechner-Bildschirm keine Fehlermeldung ausgegeben. Die muß man sich nämlich wiederum erst über den IEC-Bus-Fehlerkanal (Sekundäradresse 15) vom Laufwerks-Controller holen:

```
10OPEN 1,8,15
20INPUT# 1,A$,B$,C$,D$
30PRINT A$,B$,C$,D$
40CLOSE 1
RUN
```

Diese Anweisungsfolge darf nicht direkt (d.h. ohne Zeilennummern), sondern nur als Programm eingegeben werden, da INPUT# nicht als Direkt-Anweisung zulässig ist. Die ausgegebenen Strings haben folgende Bedeutung:

A$	Numerischer Fehlercode; 0 = kein Fehler
B$	Fehlermeldung in englischem Klartext
C$	Spurnummer, bei der der Fehler auftrat
D$	Sektor, bei dem der Fehler auftrat

Bei Basic 4.X ist das Abfragen der Fehlermeldung einfacher mit PRINT DS$ möglich. Nach der Abfrage erlischt die Fehler-LED am Laufwerk wieder.

Die Fehlercodes (im obigen Beispiel A$) lassen sich grob in folgende Gruppen einteilen:

0	kein Fehler aufgetreten
20—24, 27, 50	Fehler beim Lesen
25—29, 51, 52	Fehler beim Schreiben
30—39	Kommando unzulässig
60—67	Fehler bei OPEN
70—74	Systemfehler

Die Tatsache, daß ein laufendes Basic-Programm von einem Disk-Fehler nie abgebrochen wird, sondern dieser Fehler durch Abfrage des Floppy-Fehlerkanals stets abgefangen werden kann, ist ein gewisser Ausgleich für das Fehlen der Anweisung ON ERROR GOTO im Commodore-Basic.

Steuerkommandos

Die folgenden Anweisungen kann man dem Laufwerk innerhalb einer PRINT# -Anweisung direkt übergeben. In Basic 4.X sind sie so gewöhnlich allerdings nicht nötig, da dies der Basic-Interpreter bei Befehlen wie HEADER, COPY usw. automatisch tut.

#

Mit einem Doppelkreuz kann man einer bestimmten Sekundär-Adresse einen zugehörigen Pufferspeicher des Laufwerk-Controllers zuordnen. Folgendermaßen wird der logischen Datei 2 die Sekundäradresse 4 und der Pufferspeicher 5 zugewiesen (vgl. B-R und B-W):

```
OPEN 2,8,4,"# 5"
```

BLOCK-ALLOCATE oder B-A

Aktualisiert die Sektor-Belegungs-Liste (BAM, s.o.) und stellt fest, ob der Block belegt ist. Wenn ja, wird der nächste freie Sektor festgestellt und die Fehlermeldung NO BLOCK bereitgehalten. Beispiel für Laufwerk 1, Spur 11, Sektor 5:

```
PRINT# 1,"B-A1,11,5"
```

BLOCK-READ oder B-R

Damit ist es möglich, einen bestimmten Sektor (256-Byte-Block) von der Diskette in den Pufferspeicher des Laufwerk-Controllers zu lesen. Die folgende Anweisung liest Spur 12, Sektor 9 von Laufwerk 0 in den Pufferspeicher, der zur Sekundäradresse 4 gehört:

```
PRINT# 1,"B-R4,0,12,9"
```

BLOCK-WRITE oder B-W
Schreiben des einen Sektors aus einem Controller-Pufferbereich auf Diskette. Das folgende Beispiel schreibt den zur Sekundäradresse 4 gehörenden Puffer auf Spur 12, Sektor 9 von Laufwerk 0:

```
PRINT# 1,"B-W4,0,12,9"
```

BLOCK-EXECUTE oder B-E
Liest zunächst wie B-R einen Sektor von Diskette (gleiche Syntax), interpretiert die Daten jedoch als Maschinenprogramm und springt zu dessen Anfangsadresse. Das Programm muß mit dem Befehl RTS (hex 60) enden, damit der Floppy-Controller anschließend wieder Kommandos verarbeiten kann.

BUFFER-POINTER oder B-P
Der Pufferzeiger des Laufwerk-Controllers wird auf eine bestimmte Zeichenposition gestellt, im folgenden Beispiel auf Position 22 des zur Sekundäradresse 4 gehörenden Puffers:

```
PRINT# 1,"B-P4,22"
```

COPY oder C
Duplizieren und/oder Aneinanderhängen (Concatenation, Verkettung) von bis zu vier Quelldateien. Folgendes Beispiel kopiert die Datei PRG mit dem neuen Namen PRG1 von Laufwerk 0 nach 1:

```
OPEN 1,8,15:PRINT# 1,"C1:PRG1=C0:PRG":CLOSE 1
```

Mit folgender Kommandozeile kann man PRG1 und PRG2 aus Laufwerk 0 zu einer neuen, längeren Datei PRG in Laufwerk 1 verketten:

```
OPEN 1,8,15:PRINT# 1,"C1:PRG=C0:PRG1,C0:PRG2":CLOSE 1
```

DUPLICATE oder D
Dupliziert eine vollständige Diskette. Die Anweisung D0=1 kopiert die Disk in Laufwerk 1 auf Laufwerk 0.

INITIALIZE oder I
Lädt die „Block Availability Map" einer Diskette ins RAM; beim Einschalten oder Einlegen einer anderen Disk in ein Laufwerk erforderlich, wenn in Basic 3.X gearbeitet wird. In Basic 4.X und unter DOS 2.X ist dieser Befehl unnötig.

MEMORY-EXECUTE oder M-E
Startet ein Maschinenprogramm innerhalb des Floppy-Controller-Speichers. Nach M-E muß der niederwertige und höherwertige Adressenteil folgen. So startet man ein Programm, das im Laufwerks-RAM an der Adresse 0200 beginnt:

```
PRINT# 1,"M-E";CHR$(00);CHR$(02)
```

MEMORY-READ oder M-R
Lesen von Daten aus dem Speicher des Laufwerk-Controllers. Nach M-R müssen ohne Abstand nieder- und höherwertiger Adressenteil folgen. Die Daten kann

man dann vom Fehlerkanal lesen. Das Beispiel zeigt, wie man ein Byte von der Hex-Adresse 0100 holt:

```
10 OPEN 1,8,2:OPEN 2,8,15
20 PRINT# 1,"M-R";CHR$(00);CHR$(01)
30 GET# 2,A$:PRINT ASC(A$):CLOSE 1:CLOSE 2
```

MEMORY-WRITE oder M-W
Schreiben von Daten in den Speicher des Laufwerk-Controllers. Nach M-W müssen ohne Abstand nieder- und höherwertiger Teil der Anfangsadresse, Zahl der Bytes und die Bytes selbst folgen. Beispiel zum Schreiben der beiden Bytes 11 und 12 ab Adresse hex 0100:

```
PRINT# 1,"M-W";CHR$(00);CHR$(01);CHR$(02);CHR$(11);CHR$(12)
```

NEW oder N
Formatiert eine Diskette und löscht dabei alle vorher gespeicherten Daten. Die Disk-Kennzahl ist hinter einem Komma nach dem Disk-Namen zu schreiben. War die Disk schon einmal formatiert und möchte man die alte Kennzahl beibehalten, so kann man auch nur einen neuen Namen angeben:

```
OPEN 1,8,15:PRINT# 1,"N0:FINANZEN":CLOSE 1
```

RENAME oder R
Umbenennen einer Disk-Datei. So wird die Datei ALT in NEU umbenannt (Laufwerk 1):

```
OPEN 1,8,15:PRINT# 1,"R1:NEU=ALT":CLOSE 1
```

SCRATCH oder S
Löscht eine oder mehrere Dateien auf der Diskette. Möchte man mehrere Dateien mit ähnlichen Namen löschen, kann man die „Wildcard"-Zeichen „?" und „*" verwenden (markieren einzelne oder mehrere Zeichen des Namens als irrelevant). Beispiel zum Löschen von PRG in Laufwerk 0:

```
OPEN 1,8,15:PRINT# 1,"S0:PRG":CLOSE 1
```

USER oder U
Damit kann man zehn Sonderfunktionen innerhalb des Laufwerk-Speichers aufrufen. U1 und U2 (auch UA und UB ist zulässig!) entsprechen B-R und B-W. Die übrigen führen Sprungbefehle aus (Adressen hexadezimal angegeben):

U3	1300	U6	D008
U4	1303	U7	D00B
U5	1306	U8	D00E
U:	E18E (Reset)	U9	D0D5

VALIDATE oder V
Entspricht der Anweisung COLLECT in Basic 4.X: Alle nicht korrekt geschlossenen Dateien werden im Inhaltsverzeichnis gelöscht und die von ihnen belegten Spuren in der BAM (s.o.) wieder freigegeben. Tritt dabei ein Lesefehler auf, so bleibt die Diskette dagegen unverändert.

Schreiben und Lesen von Textdateien

Um einen String unter dem Dateinamen TEST auf Diskette zu speichern, ist eine Anweisungsfolge wie diese notwendig:

```
10 OPEN 1,8,2,"0:TEST,SEQ,WRITE"
20 PRINT# 1,"Dies ist ein Test-String"
30 CLOSE 1
RUN
```

Die OPEN-Anweisung öffnet den logischen Kanal (die Dateinummer) 1 auf der IEC-Bus-Adresse 8 und spricht Laufwerk 0 an. Die Sekundäradresse ist gleichgültig, solange sie nicht 15 ist (15 ist dem Kommando- und Fehlerkanal vorbehalten); sie bestimmt lediglich, welcher Pufferspeicher-Bereich innerhalb des Laufwerks verwendet wird.

Dieses kurze Programm beschwert sich mit einer blinkenden Laufwerks-LED, wenn auf der Diskette bereits eine Datei namens TEST existiert. Will man das vermeiden und nimmt in Kauf, daß bestehende Dateien ohne Warnung überschrieben werden, so kann man in Zeile 10 vor die Laufwerks-Nummer einen „Klammeraffen" stellen:

```
10 OPEN 1,8,2,"@0:TEST,S,W"
```

Ferner wurde hier gezeigt, daß man SEQ und WRITE auch abkürzen kann. S gibt den Dateityp an:

P = Programm-Datei (Basic- oder Maschinenprogramm)
S = sequentielle Datei (wird stets ab dem Anfang gelesen)
U = Random-Access-Datei (Datei für wahlfreien Zugriff)

WRITE oder READ (bzw. W oder R) teilen dem Laufwerk mit, ob geschrieben oder gelesen werden soll.

Um den String wieder zurückzulesen und auf dem Bildschirm darzustellen, kann man das folgende Mini-Programm zur Demonstration verwenden:

```
10 OPEN 1,8,2,"0:TEST,S,R"
20 INPUT# 1,E$:PRINT E$
30 CLOSE 1
RUN
```

Wie man im Falle einer blinkenden Laufwerks-LED eine eventuelle Fehlermeldung vom Laufwerk holt und sie anzeigt, wurde ja bereits weiter oben beschrieben. Die hierzu nötige Routine kann man am Schluß des Mini-Programms anhängen.

In Commodore-Basic gibt es wie beim Apple-II keinen Befehl LINE INPUT, der es gestatten würde, Zeilen mit Kommata und Doppelpunkten nicht nur von der Tastatur, sondern auch von der Diskette zu lesen. Eine Lösung dieses Problems wurde bereits im Kapitel über Apple-DOS 3.3 vorgeschlagen, nämlich, Strings schon beim Abspeichern auf Diskette mit Anführungszeichen zu umrah-

men. Dieses Prinzip funktioniert bei Commodore-Rechnern ebenso und braucht daher hier nicht nochmals dargestellt zu werden.

Möchte man z.B. zum Senden eines Basic-Programms per Telefonmodem dieses als ASCII-Textdatei auf Diskette bringen und nicht in Form von Token-Codes statt der Befehlsworte, so kann man dies mit der Anweisung CMD erreichen, indem man folgende zwei Befehlszeilen von Hand eintippt:

```
OPEN 1,8,2,"0:TEST,S,W":CMD 1:LIST
PRINT# 1:CLOSE 1
```

Leider gibt es keinen so eleganten Weg wie den EXEC-Befehl von Apple-DOS, um eine solche ASCII-Datei wieder in ein lauffähiges Basic-Programm zurückzuverwandeln. Das Problem ist nur lösbar, indem man mit GET# Zeichen für Zeichen von der Disk-Datei holt, mit POKE in den Tastatur-Puffer bringt und so eine manuelle Eingabe simuliert. Darauf sei hier jedoch nicht näher eingegangen, weil die nötigen Adressen leider bei jedem Commodore-Computer anders lauten und man nur hoffen kann, daß spätere Geräte dieses Herstellers einfachere Möglichkeiten dazu anbieten.

5 Anwender-Software

Unter Anwender-Software versteht man Programme, die unter Verwendung der Systemsoftware (Betriebssystem, Interpreter usw.) ein Anwenderproblem lösen. Beispiele dafür sind Textverarbeitungs-, Kalkulations- oder Datenbank-Programme.

5.1 Anforderungen an Anwender-Software

5.1.1 Ergonomie

Ergonomie ist eine Wissenschaft, deren Zweck es ist, zu ergründen, wie Gebrauchsgegenstände oder Verfahren aussehen müssen, um ihrem Bestimmungszweck möglichst benutzerfreundlich gerecht zu werden. Bei Computer-Hardware umfaßt sie beispielsweise die Konstruktion der Tastatur (leicht gewölbte Anordnung, leichtgängige Tasten, niedrige Bauhöhe), des Bildschirms (augenschonende Farbe wie z.B. grün, ausreichende Nachleuchtdauer für Flimmerfreiheit, entspiegelte Oberfläche) oder auch den Aufbau der Zentraleinheit (kleines Gehäuse, das auf dem Arbeitstisch wenig Platz einnimmt; Zeichengenerator mit möglichst vielen Matrixpunkten pro Zeichen für gute Lesbarkeit usw.).

Bei Software versteht man unter Ergonomie ebenfalls, daß sich der Computer möglichst benutzerfreundlich verhält. Zunächst einmal ist der Benutzer natürlich der Programmierer, und er wird wegen seiner täglichen Übung mit dem Rechner vermutlich teilweise andere Dinge unter Ergonomie verstehen als der spätere, gewöhnlich weit weniger technisch vorgebildete Anwender des fertig entwickelten Programms.

Abb. 5.1.1.1 macht unterschiedliche Möglichkeiten der Programm-Entwicklung deutlich. Zu jedem Punkt sind zwei konträre Alternativen genannt, wobei es für jede durchaus gute Gründe gibt. Der Programmierer muß nun abwägen, welche Eigenschaften der beiden Alternativen für das jeweilige Anwenderproblem von größerer Bedeutung sind.

Geschwindigkeit und Speicherplatz-Bedarf

Beispielsweise ist bei der Entwicklung von Programmen für sehr zeitkritische Aufgaben (Regelungstechnik, schnell bewegte Grafiken) vor allem eine hohe Geschwindigkeit wichtig, so daß höhere Programmiersprachen wie Basic und Pascal

Nr.	Methode	Vorteile dieser Methode
1a	Höhere Sprachen	Später leicht änder- und korrigierbar
1b	Assembler	Geringer Speicherbedarf, hohe Geschwindigkeit, einfacher Programmschutz
2a	Integrierte Software	Einheitliche, leicht erlernbare Benutzer-Oberfläche; Dateien aller Teilprogramme kompatibel
2b	Getrennte Programme	(Ver-)Kauf auch einzeln möglich; leistungsfähigere Einzelprogramme; Entwicklung leicht von mehreren Programmierern möglich
3a	Strukturiertes, gut kommentiertes Programm	Später leicht korrigier- und änderbar; einzelne Routinen auch für spätere Programmieraufgaben verwendbar
3b	Trickreicher Kompaktcode	Geringer Speicherbedarf; hohe Geschwindigkeit; urheberrechtlich und technisch schützbar
4a	Wenige Kommandos	Kurzes Programm, geringer Lernaufwand beim Benutzer
4b	Viele Kommandos	Hoher Programmkomfort, große Flexibilität
5a	Kommandoeingabe mit Menüs	Im Idealfall kein Handbuch mehr erforderlich, geringster Lernaufwand
5b	Kommandoeingabe m. Worten	Schnelles und effizientes Arbeiten, große Zahl von Befehlsworten mögl.
6a	Häufige Sicherh.-Abfragen	Kein versehentliches Ändern und Löschen von Daten (z.B."Löschen:j/n?")
6b	Wenig Sicherheitsabfragen	Schnelleres Arbeiten möglich, wenn man sowieso weiß, was man will
7a	Fehlermeldungen im Klartext	Ohne Blättern im Handbuch sofort verständlich
7b	Fehlermeldung als Code	Geringer Speicherbedarf, deshalb viele differenzierte Meldungen möglich

Abb. 5.1.1.1
Überlegungen vor dem Schreiben eines Programms

8a	Bildschirm-orientierte Ein-/Ausgabe	Übersichtliche Darstellung, sichere Eingabe z. B. mit Masken, einfaches Editieren mit dem Cursor
8b	Zeilen-orientierte Ein-/Ausgabe	Geringer Programm-Aufwand, auch mit externen Terminals ohne Steuerzeichen-Normung möglich
9a	Eingabe mit Maus, Lichtgriffel, Joystick u.ä.	Einfache Steuerung von Cursor und Bewegungsabläufen, bequeme Eingabe von Grafiken
9b	Eingabe nur mit Tastatur	Bequeme Eingabe von Texten, numerischen Daten und Grafik-Koordinaten
10a	Datei-Bearbeitung auf Disk	Dateien können nahezu beliebig lang sein, RAM vollständig für das Programm verfügbar, kein Verlust der geänderten Daten z.B. bei "Absturz"
10b	Datei-Bearbeitung im RAM	Hohe Geschwindigkeit bei häufigen Änderungen (z. B. Text editieren)
11a	Einzelne Programmteile werden nachgeladen	Mehr Platz im RAM für Variablen bzw. Daten verfügbar; fast beliebig umfangreiche Software möglich (Overlays)
11b	Gesamtprogramm im RAM/ROM	Schneller Wechsel zwischen einzelnen Programmteilen möglich
12a	Programm im RAM	Kein Adressenraum-Bedarf bei Nichtbenutzung; Programm muß nicht in Maschinensprache vorliegen
12b	Programm im ROM/EPROM	Ev. kein Disk-Laufwerk erforderlich (bei tragbaren Rechnern von Vorteil); Programm nach dem Einschalten sofort verfügbar

zu Abb. 5.1.1.1

trotz der vielen Vorteile, die sie bieten, praktisch nicht in Frage kommen. Auch die Verwendung von Compilern, die echten Maschinencode liefern, löst dieses Problem nur für manche Anwendungen, da der compilierte Objektcode noch nicht so schnell sein kann wie ein von Haus aus in Assembler geschriebenes Programm, in

das der Programmierer sein gesamtes Können investiert hat (so schlau wie ein guter Programmierer ist leider kein noch so schlau geschriebener Compiler!). Ein weiterer Grund, warum man immer noch Assembler-Programme schreibt, ist der, daß bei preiswerten Einplatinen-Computern oder Einchip-Computern (z.B. zur Steuerung einer Waschmaschine oder eines Telefon-Wählautomaten) relativ wenig Speicherplatz zur Verfügung steht, der nur mit einem Maschinenprogramm optimal genutzt werden kann.

Mit Ergonomie hat es aber auch zu tun, wie lange der Benutzer auf eine Antwort des Rechners bzw. auf irgendwelche Ergebnisse warten muß. Rechenintensive Programme, etwa in der Astronomie oder bei Iterations-Aufgaben, können bei Programmierung mit einem Basic-Interpreter einige Minuten benötigen. Bei geschickter Assembler-Programmierung schrumpft der Zeitbedarf auf wenige Sekunden, wenn man z.B. eigene Rechenroutinen schreibt, die jeweils nur gerade so genau sind, wie es das Teilproblem erfordert und deshalb schneller sein können als mit der oft unnötig hohen Genauigkeit der Basic-Routinen.

Integrierte Software

Die meisten Computeranwender setzen ihr Gerät nicht nur für eine einzige, sondern für mehrere unterschiedliche Aufgaben ein, so etwa zur Textverarbeitung, zur Umsatz-Kalkulation, zum Erstellen von Umsatz-Grafiken, zum Führen von Dateien als elektronischem Kartei-Ersatz und zur Kommunikation mit anderen Rechnern z.B. über ein Telefon-Modem.

Selbstverständlich gibt es für jede einzelne der genannten Aufgaben gute Programme. Der Nachteil solcher getrennter Programme zeigt sich aber schnell, wenn man zum Beispiel versucht, die mit dem Kalkulations-Programm erstellte Umsatz-Tabelle in einen mit dem Editor erstellten Text einzubauen und das Resultat dann mit dem Kommunikationsprogramm an eine Außenstelle zu senden: Die Programme verwenden zum Teil unterschiedliche Steuerzeichen, das Dateiformat ist nicht kompatibel.

Einen Ausweg aus diesem Problem stellt die sogenannte „integrierte Software" dar. Dabei handelt es sich im Prinzip um ein Programmpaket, dessen einzelne Teile untereinander vollständig kompatible Dateiformate verwenden. Im Idealfall sind wesentliche Benutzer-Kommandos auch in allen Teilprogrammen gleich aufgebaut (z.B. ESC für Programm verlassen, CTRL-C für Ende der Eingabe, CTRL-S für Textausgabe anhalten usw.). Dadurch ist ein integriertes Software-Paket in der Regel bedienungsfreundlicher als eine Anzahl einzelner Programme mit völlig unterschiedlichen Kommandos.

Auf der anderen Seite ist allerdings festzustellen, daß die meisten auf dem Markt verfügbaren integrierten Software-Pakete gegenüber den sonst erforderlichen vielen Einzelprogrammen zwar preiswerter sind, aber in vielen Details (z.B. beim Editieren von Texten) auch weniger leistungsfähig sind.

```
CHOOSE ONE OF THE FOLLOWING OPTIONS


        <1>   COPY FILES
        <2>   CATALOG
        <3>   SPACE ON DISK
        <4>   UNLOCK FILES
        <5>   LOCK FILES
        <6>   DELETE FILES
        <7>   RESET SLOT & DRIVE
        <8>   VERIFY FILES
        <9>   QUIT

WHICH WOULD YOU LIKE? 1
COPY FILES
SOURCE SLOT?6
      DRIVE?1

DESTINATION SLOT?6
            DRIVE?2


FILENAME?=
DO YOU WANT PROMPTING? N
INSERT DISKS.
```

Abb. 5.1.1.2
Menü des Apple-DOS-Hilfsprogramms FID

Menü oder Kommando-Eingabe?

Wenn man ein neu erworbenes Programm zum ersten Mal benutzt, ist man einen erheblichen Teil der Zeit noch damit beschäftigt, im zugehörigen Handbuch nachzulesen, mit welchem Kommando die nächste Aktion durchgeführt werden kann. Man wünscht sich dann ein Programm, das so ergonomisch aufgebaut ist, daß man dazu überhaupt kein Handbuch braucht.

Diesem Idealbild kommen menügesteuerte Programme recht nahe. Dabei werden alle im jeweiligen Betriebszustand möglichen Kommandos in einer Liste auf dem Bildschirm angezeigt. *Abb. 5.1.1.2* zeigt das Menü, mit dem sich das Apple-II-Hilfsprogramm FID (siehe Apple-DOS 3.3) beim Start meldet. Den gewünschten Punkt kann man durch Eingabe der zugehörigen Ziffer und anschließendes Drükken von Return auswählen.

Typisch für menügesteuerte Programme sind auch die zahlreichen Infobox-Dienste (siehe Kapitel über Modems). Wenn man einen solchen Dienst per Akustikkoppler anruft, besitzt man ja zu ihm keinerlei schriftliche Bedienungsanleitung. Deshalb erhält man, sobald die Verbindung hergestellt ist, erst einmal ein Menü auf dem Bildschirm, aus dem man sich den gewünschten Punkt aussuchen kann.

Nachteilig ist bei Menüs gegenüber der Eingabe auswendig gelernter Kommandos, daß für das Menü jedesmal ein Teil des Bildschirms belegt wird. Eine Lösung

dieses Problems ist, das Menü per Tastendruck (z.B. ESC) ausblenden zu können (z.B. bei Wordstar realisiert).

Das Handbuch zum Computer

Daß gute, verständliche und doch korrekte Handbücher, die auch zu selteneren Detailfragen Auskunft geben, keineswegs selbstverständlich sind, dürfte fast jedem Leser bewußt sein, der selbst einen Computer besitzt. Auf deren Problematik wird absichtlich in diesem Kapitel eingegangen, weil sie im weiteren Sinne ebenfalls als Software bezeichnet werden können und zur Ergonomie des Systems wesentlich beitragen. (Was nützen die schönsten Kommandos und ROM-Routinen, wenn sie nicht verständlich und ausführlich erläutert werden?) Im leider selten vorkommenden Idealfall gliedert sich ein selbstverständlich in deutscher Sprache geschriebenes Computer-Handbuch (Manual) grob in folgende Teile:

1. Auspacken und Konfigurieren des Rechners
2. So nehme ich die Kiste sofort (!) mit einem Demo-Programm in Betrieb
3. Empfehlungen für den Umgang mit der Hardware, technische Daten

Falsch	Richtig	Bemerkung
19,2 KBit/s	19,2 kBit/s	Für 19200 Bit/s; K=1024 und k=1000!
2 CPU's	zwei CPUs	Apostroph nur bei englischem Genitiv
64 K RAM	64 KByte RAM	K ist eine Temperatur-Einheit (Kelvin)
64 KB RAM	64 KByte RAM	KB könnte KByte oder KBit heißen
64 KBit Chip	64-KBit-Chip	Gilt als ein Wort
8 Pin-Stecker	8-Pin-Stecker	Es sind nicht acht Stecker gemeint!
96 polig	96polig	Gilt als ein Wort
Applecomputer	Apple-Computer	Namen sind abzutrennen
ASCII-Code	ASCII	Das C in ASCII heißt bereits Code
Autostartrom	Autostart-ROM	ROM ist der Name für Read-Only Memory
BASF Diskette	BASF-Diskette	Gilt als ein Wort
Basicprogramm	Basic-Programm	Namen sind abzutrennen
Centronix	Centronics	Firmen- und Schnittstellen-Name
Controler	Controller	Verdoppelung des l bei Mehrzahl
CP/M System	CP/M-System	Gilt als ein Wort
Die 6809 CPU	Die CPU 6809	Wortstellung nur im Englischen üblich
ECB-Bus	EC-Bus; ECB	Das B in ECB heißt bereits Bus
In 1985	Im Jahr 1985	Angelsächsischer Ausdruck
Graphic	Grafik	Manchmal noch mit ph geschrieben
LCD-Display	LC-Display	Das D in LCD bedeutet bereits Display
LED-Diode	LED	Das D in LED bedeutet bereits Diode
Microcomputer	Mikrocomputer	Im Deutschen mit k
ROM resident	ROM-resident	Gilt als ein Wort
Standart-Code	Standard-Code	Eine Standarte ist eine Flagge
V24	V.24	CCITT-Schnittstellen-Norm
Zilog Z80 CPU	Zilog-Z80-CPU	oder: CPU Z80 von Zilog

Abb. 5.1.1.3 Schreibweisen von Fachausdrücken der Computertechnik

4. Kommandos des Disketten-Betriebssystems (falls vorhanden), Bedeutung der (englischen?) Fehlermeldungen
5. Befehle der eingebauten höheren Programmiersprache, Bedeutung der (englischen?) Fehlermeldungen
6. Einige typische Beispielprogramme (z. B. in Basic)
7. Programmierung in Maschinensprache (auch Speicherbelegung usw.)
8. Hinweise zur Wartung (Service-Vertretungen des Herstellers usw.)
9. Stichwort-Verzeichnis mit Befehlen und Fachausdrücken

Es lohnt sich übrigens, beim Kauf eines Computers nicht nur auf die Hardware, sondern auch auf die Qualität des mitgelieferten Handbuchs (es können auch mehrere sein) zu achten. Ob es wirklich in Deutsch geschrieben ist, können Sie durch Vergleichen mit der Tabelle feststellen, die in *Abb. 5.1.1.3* abgedruckt ist — der Schein trügt manchmal!

5.1.2 Kosten und Portabilität

Normalerweise versucht man schon bei der Entwicklung, Software so zu gestalten, daß sie mit möglichst geringem Arbeitsaufwand auch für andere Computertypen als den ursprünglich vorgesehenen angepaßt werden kann. Nur so ist es möglich, ausreichende Verkaufs-Stückzahlen sicherzustellen und somit einen konkurrenzfähigen Preis zu erzielen. Denn da man die Kosten von Datenträgern und dem Papier der Bedienungsanleitung im Vergleich zu marktüblichen Software-Verkaufspreisen nahezu vernachlässigen kann, ist der Preis des fertigen Produkts in erster Näherung der zu erwartenden Verkaufsstückzahl umgekehrt proportional.

Entwicklungskosten

Dazu gleich ein praktisches Beispiel. Ein Programmierer ist mit der Entwicklung eines Programms zur Datenfernübertragung (DFÜ) inklusive abschließender Funktionstests und dem Erstellen der Bedienungsanleitung einen Monat befaßt. Das Unternehmen, bei dem er angestellt ist, muß für diese Entwicklung folgende Kosten ansetzen:

1. Brutto-Gehalt und Arbeitgeber-Anteile von Sozialbeiträgen
2. Anteilige Mietkosten für den Arbeitsplatz
3. Umgelegte Investitionskosten (Möbel, Computer, Drucker usw.)
4. Laufende Kosten (Telefongebühren, Verbrauchsmaterial usw.)

Für diese Kosten setzen wir in unserem Beispiel pauschal 10000 DM an. (Das ist nicht einmal besonders viel.) Jetzt kommt ein etwas schwieriger Punkt, nämlich die Abschätzung der zu erwartenden Stückzahl. Da nicht einmal Programmierer hellsehen können und Kaufleute schon gar nicht, kann man nur hoffen, durch gezielte Werbemaßnahmen die potentiellen Anwender anzusprechen. Leider beißt sich die Katze bei diesem Punkt in den Schwanz: Eine geringe angenommene Stückzahl führt zu einem hohen Einzelstückpreis, wodurch man wiederum eher weniger als mehr verkauft. Zur Abschätzung der Verkaufsstückzahl gehört also gerade bei Software, wo diese Zahl so direkt wie kaum sonst die Preisgestaltung beeinflußt, eine gehörige Portion Marktkenntnis, aber auch etwas Glück. Für unser Beispiel schätzen wir die Stückzahl, mit der sich diese erste Programmversion (ohne weitere Überarbeitungs-Kosten) verkaufen läßt, auf 1000.

Distributionskosten

Leider läßt sich aber damit noch lange nicht der tatsächlich sinnvolle Verkaufspreis des Produkts ermitteln. Weitere Kosten sind nämlich:

5. Kosten für Werbemaßnahmen. Für eine einzige Anzeigenseite einer Fachzeitschrift muß man je nach Auflage 2000...6000 DM hinlegen, und in die Köpfe der Leser kann man sich nur durch ständige Wiederholung bohren. Auch Prospekte gehören hierzu.
6. Kosten für Händlerinformation. Da ein Direkt-Vertrieb selten in Frage kommt, muß man ein flächendeckendes Netz von Händlern mit Informationen über das neue Produkt versorgen. Dazu können z.B. auch Reisekosten zählen, um die Händler zu besuchen.
7. Der Handelsrabatt, zu dem Händler das Produkt beziehen können. Er beträgt bei Software zwischen 30 und 60 Prozent des Endverkaufspreises, je nachdem, wie beratungsintensiv die Software ist.
8. Last but not least will man an der Software auch noch etwas verdienen, hierfür sind etwa 10% vom Endverkaufspreis als Gewinnanteil ein grober Anhaltspunkt.

Für unser konkretes Beispiel ergibt sich dann zunächst etwa folgende Aufstellung für die Punkte 1 bis 6:

1....4.	10000 DM	(Entwicklungskosten)
5.	6000 DM	(4mal ¼ Anzeigenseite und 2000 Prospekte)
6.	4000 DM	(Händler-Info und Reisekosten)

Das ergibt zusammen runde 20000 DM. Teilt man diese Summe durch die Stückzahl 1000, erhält man zunächst Kosten von 20 DM pro Stück, mit 5 DM Gewinnanteil (ergibt 25 DM) und unter Berücksichtigung von 50% Händler-Rabatt 50 DM. Dazu muß man noch die Datenträger- und Kopierkosten sowie den Druck des Anleitungs-Heftchens rechnen, zusammen etwa 10 DM. Mit 14% Mehrwertsteuer kann man das Programm also für 68,40 DM unters Volk bringen — voraus-

gesetzt, es werden etwa innerhalb eines Jahres wirklich wie angenommen 1000 Stück davon verkauft. Nach dieser Zeit haben sich die Entwicklungskosten amortisiert, der Gewinnanteil steigt.

Meist stellt man kurz nach Verkaufsbeginn (die Kunden sind die beste Endkontrolle ...) noch kleinere Unschönheiten oder „Bugs" im Programm fest, die einer Überarbeitung bedürfen, so daß eine verbesserte Version auf den Markt kommt. Bei teureren Programmen enthält der Kaufpreis oft zusätzlich einen Anteil zur späteren kostenlosen Nachlieferung einer verbesserten Version.

Bei den meisten Programmen, selbst solchen, die mit einem sogenannten Kopierschutz versehen sind (dazu später, jeder ist irgendwann geknackt), stellt man oft schon nach einigen Monaten schon deshalb ein Abflauen der Verkaufsstatistik fest, weil irgendeine „geknackte" Version auf dem „grauen" Markt ihr Unwesen treibt. Ferner soll es durchaus Händler geben, die (um die Hardware besser zu verkaufen) das eine oder andere Programm gratis dazuliefern — allerdings ohne Handbuch. Diese für den Software-Hersteller unerwünschten Distributionskanäle etablieren sich um so mehr, je weniger ein Handbuch erforderlich ist; gerade ergonomisch mit Menüs unterstützte Programme sind hiervon betroffen.

Portabilität

Für die Übertragbarkeit eines Programms auf andere Rechnertypen gibt es mehrere Stufen:

1. Physikalische Kompatibilität: Die Diskette für Rechnertyp A ist unverändert für Typ B geeignet.
2. Code-Kompatibilität: Zwar ist das Disk-Aufzeichnungsformat nicht kompatibel, im Programm selbst sind aber keine Änderungen nötig.
3. Code-Portabilität: Nur geringe Änderungen im Quellprogramm sind für das Übertragen auf einen anderen Rechner erforderlich.
4. Struktur-Portabilität: Der gleiche logische Programmablauf eignet sich auch für andere Rechner, das Quellprogramm muß jedoch unter Zuhilfenahme desselben Flußdiagramms oder Struktogramms neu geschrieben werden.
5. Nicht portabel: Ein anderer Rechner wäre wegen seiner Hardware-Voraussetzungen für die Lösung des gegebenen Problems ohnehin ungeeignet.

Fall 1 ist natürlich nur bei weitgehend kompatiblen Rechnern zu erwarten, so z.B. bei der Familie Apple-II-kompatibler Computer oder bei IBM-PC-kompatiblen Geräten. Nicht in jedem Fall garantiert die physikalische Kompatibilität allerdings auch die unveränderte Lauffähigkeit des Programms. So können z.B. IBM-PC, IBM-AT und Olivetti-M24 zwar ohne weiteres das gleiche Diskettenformat lesen, manchmal sind aber beim Ansprechen bestimmter Hardware-Adressen Programmänderungen nötig. Ein typisches Beispiel für Fall 1 ist jedoch die direkte Verwendbarkeit von Compaq-Disketten für den IBM-PC.

Fall 2 ist recht häufig, da es oft für ein und denselben Rechnertyp unterschiedliche Möglichkeiten der Floppy-Laufwerksversion gibt (z.B. ein- oder doppelseitig, 8 oder 9 Sektoren pro Spur usw.). Ein Beispiel hierfür wäre auch die Verwendung eines für den IBM-PC geschriebenen Programms auf dem Victor/Sirius-1.

Fall 3 tritt immer auf, wenn bei einem Assemblerprogramm geringe Änderungen nötig sind, da zwei Rechner zwar mit dem gleichen oder einem objektcodekompatiblen Prozessortyp arbeiten, aber eine unterschiedliche Speicherbelegung oder Hardware-Konfiguration besitzen. Besonders hilfreich ist hier die Verwendung eines Assemblers, der bedingte Anweisungen erlaubt (IF, THEN, ELSE). Dann kann man dafür sorgen, daß durch Setzen eines einzigen Parameters am Programm-Beginn der gewünschte Objekt-Rechnertyp festgelegt wird.

Eine Code-Kompatibilität ist aber meist auch gegeben, wenn das Programm nicht in Assembler, sondern einer höheren Programmiersprache geschrieben ist, die im gleichen „Dialekt", z.B. GW-Basic, auf mehreren Rechnertypen verfügbar ist. Eventuell nötige geringfügige Anpassungen sind auch in Basic besonders leicht durchzuführen, wenn man alle systemspezifischen Parameter (z.B. Adressenangaben für PEEK- und POKE-Befehle, Cursor-Steuerzeichen usw.) ähnlich wie beim Vereinbarungs-Block in Pascal gleich zu Beginn des Programms als Variablen (genauer gesagt, als Konstanten mit Variablennamen) definiert.

Fall 4 ist typisch für ein Assembler-Programm, das z.B. für die CPU Z80 geschrieben ist und nun auf einen 68000-Rechner transferiert werden soll. In diesem konkreten Fall gibt es allerdings (wenn nicht auf eine sehr hohe Geschwindigkeit Wert gelegt wird) einen eleganten Ausweg in Form eines Interpreter-Programms für den Prozessor 68000, der die Z80-Maschinenbefehle Byte für Byte interpretiert. Das so emulierte System entspricht etwa einem 2-MHz-Z80. In anderen Fällen gibt es Hilfsprogramme, die z.B. 8080-Quellprogramme in 8086-Objektcode übersetzen. Wenn auch diese Möglichkeit nicht existiert, muß man in den sauren Apfel beißen und das Programm neu schreiben. Dafür ist natürlich ein vorher erstelltes Struktogramm oder Flußdiagramm sehr hilfreich. Geringe Änderungen im Ablauf sind bei einem stark abweichenden CPU-Befehlssatz u.U. jedoch dennoch erforderlich, z.B. weil die Flags im Status-Register beim 6502 schon von allen Transfer-Befehlen gesetzt werden, beim Z80 jedoch nicht.

Fall 5 kann gegeben sein, wenn man beispielsweise ein Programm, das unbedingt hochauflösende Grafik als Ausgabemedium erfordert, auf einen Rechner transferieren möchte, der diese Hardware-Eigenschaft gar nicht besitzt. Meist gibt es aber doch irgendwelche Auswege, z.B. hier die Verwendung eines grafikfähigen Druckers oder eines Plotters als Ausgabemedium.

5.1.3 Schutz vor Kopierern

Software-Entwicklung ist teuer und ist daher nicht mit dem Preis einer Leerdiskette abgegolten. Das wurde im letzten Kapitel an einem Kalkulationsbeispiel deutlich; es wurde aber auch darauf hingewiesen, daß viele Programme nach wenigen Monaten plötzlich auf dem „grauen" Markt auftauchen und als Tauschobjekte dienen. Schlimmer noch, manche Leute verdienen sich die Marmelade für ihre Brötchen nicht etwa mit eigenem Hirnschmalz, sondern mit geklauten Programmen, die sie per Kleinanzeige an den Mann bringen — natürlich (wenn überhaupt) mit der Kopie einer Kopie des Handbuchs und mit keinerlei späterer Unterstützung.

Der Software-Hersteller hat jedoch zwei Möglichkeiten in der Hand, gegen ein solches illegales Kopieren wirksam vorzugehen, nämlich zum einen den rechtlichen Schutz seiner Programme und zum zweiten technische Verfahren, um das Kopieren so gut wie unmöglich zu machen. Es sei gleich vorangestellt, daß man sich allein von der rechtlichen Absicherung nicht allzu viel erwarten darf: Wenn z.B. zwei gute Bekannte sich gegenseitig teure Programme kopieren, wird das so schnell kein Dritter erfahren — wo kein Kläger, da kein Richter. (Auch das Abhören von Funktelefon-Gesprächen ist nicht unmöglich, nur weil es verboten ist!) Wo allerdings eine zwielichtige Firma schwarz kopierte Programme unlizensiert per Zeitschriften-Anzeigen feilbietet, sind Rechtsmittel sehr wohl geeignet.

Kaufverträge

Bei wertvollen Programmen, die einige hundert Mark kosten, wird beim Kauf normalerweise ein Vertrag geschlossen, der das Kopieren nur für den persönlichen Bedarf zuläßt, so etwa bei den meisten Betriebssystemen (CP/M, MS-DOS usw.), Interpretern und Compilern. Ein technischer Kopierschutz wäre hier nicht sinnvoll, da es dem Käufer unbedingt möglich sein muß, z.B. mehrere Disketten mit dem Betriebssystem zu formatieren.

„Für den persönlichen Gebrauch" bezieht sich im Kaufvertrag nicht auf juristische Personen wie z.B. ganze Firmen. Wenn ein Unternehmen mehrere Arbeitsplätze mit einem bestimmten Programm ausstatten möchte, muß das Programm auch entsprechend oft gekauft werden. Aus diesem Grund werden auch üblicherweise die Handbücher zu Programmen nicht einzeln angeboten. (Natürlich erhält man vom Software-Entwickler einen erheblichen Mengenrabatt, wenn man ein Programm z.B. gleich zehnmal kauft.)

Bei Compilern sieht der Kaufvertrag meist vor, daß das sogenannte Laufzeitpaket (Run Time Unit) frei von Lizenzvereinbarungen ist und nur der eigentliche Übersetzer selbst nicht weitergegeben werden darf. Dadurch ist es z.B. einem Software-Entwickler möglich, compilierte Programme auf Disketten zu verkaufen, die notwendigerweise auch das Laufzeitpaket enthalten.

Als Kunde sollte man bei Kaufverträgen auf jene Klauseln achten, die die Nachlieferung späterer, verbesserter Programmversionen einschließen. Seriöse Softwa-

re-Häuser berechnen dafür gewöhnlich nur die Datenträger- und Versandkosten oder bieten gegen Gebühr einen langfristigen „Updating Service" (Wartungsvertrag) an.

Urheberrechtlicher Schutz

Nach der Definition der internationalen „Weltorganisation für geistiges Eigentum" versteht man unter Computer-Software eine „Folge von Befehlen, die nach Aufnahme in einen maschinenlesbaren Träger fähig sind, zu bewirken, daß eine Maschine mit informationsverarbeitenden Fähigkeiten eine bestimmte Funktion oder Aufgabe oder ein bestimmtes Ergebnis anzeigt, ausführt oder erzielt" (GRUR International 1978, S. 286 ff.).

Eine Programm-Beschreibung (Listing, Benutzungsanweisung, Ablaufdiagramm) ist in Ergänzung dazu eine „vollständige prozedurale Darstellung in sprachlicher, schematischer oder anderer Form, deren Angaben ausreichend sind, um eine Folge von Befehlen festzulegen, die ein ihr entsprechendes Computerprogramm darstellen".

Voraussetzung für die Möglichkeit eines urheberrechtlichen Schutzes ist grundsätzlich, daß es sich um eine „persönliche geistige Schöpfung" handelt. Die Beurteilung dieser Frage liegt zum Teil in der Ermessensfrage der damit befaßten Gerichte. In der Tat sind hierzu sehr unterschiedliche Urteile ergangen. Ein Programm ist jedoch nur dann überhaupt urheberrechtlich schützbar, wenn folgende Voraussetzungen gegeben sind:

1. Geistige Leistung: Analytisches und planerisch-konstruktives Denken gehören hierzu (GRUR 1982, S. 453).
2. Formgebung. Es genügt nicht, wenn der Entwickler seine Gedanken für sich behält; er muß sie irgendwie zu Papier oder auf einen Datenträger gebracht haben. Theoretisch genügt hierfür auch bereits z. B. ein Flußdiagramm.
3. Individualität. Diese ist nur gegeben, wenn das Ergebnis der Problemlösung nicht exakt vorhersagbar ist (das ist nahezu bei jedem Programm gegeben, das über eine einfache arithmetische Formel hinausgeht).

Wenn alle drei Punkte gegeben sind, ist allein der Urheber nach § 11 des Urheberrechtsgesetzes (UrhG) zur Nutzung des Programms, d. h. zur Weitergabe an Dritte, berechtigt. Gibt er es an ein Softwarehaus zur Vermarktung weiter, kann er verlangen, daß sein Name als Urheber genannt wird und eine wie immer geartete Entstellung des Werks verbieten.

Gemäß § 16 UrhG hat der Urheber das Recht, entweder selbst Kopien seines Werks herzustellen oder es vervielfältigen zu lassen und dann zu verbreiten. Dazu gehört auch eine „Übertragung auf Vorrichtungen zur wiederholbaren Wiedergabe", z. B. per Telefonmodem auf einen anderen Computer. § 97 regelt Schadensersatz und Unterlassungs-Ansprüche des Urhebers, wenn dieser feststellt, daß sein Werk von Dritten illegal verbreitet wird.

```
10PRINT"DAS IST NUR EIN"
20PRINT"KLEINES TESTPROGRAMM"
100REM (C) H.CRACKER

>CALL-151

*801.804

0801- 18 08 0A 00
*818.81B

0818- 34 08 14 00
*834.837

0834- 48 08 64 00
*836:FF FF

*

>LIST

10PRINT"DAS IST NUR EIN"
20PRINT"KLEINES TESTPROGRAMM"
65535REM (C) H.CRACKER

>65535REM (C) M.HACKER

?SYNTAX ERROR
```

Abb. 5.1.3.1
Die hier auf einem Apple-II von 100 auf 65535 geänderte Zeilennummer kann von Basic aus nicht mehr ohne weiteres gelöscht oder geändert werden

Schutz vor Programmänderungen

Manche Programme sind so aufgebaut, daß sie nicht gelistet und auch nicht verändert werden können. Würde nämlich jemand das Programm so sehr verändern, daß es in weiten Teilen kaum noch an das Original erinnert, so würde es nicht mehr unter den urheberrechtlichen Schutz des ursprünglichen Autors fallen. Ferner will man oft vermeiden, daß ein Urheberrechts-Vermerk, der beim Starten des Programms auf dem Bildschirm erscheint, gelöscht oder verändert wird.

Ein einfaches Beispiel zum Anlegen eines Urheberrechts-Vermerks in einem Basic-Programm ist folgendes, hier für den Apple-II (die Adressen lauten beim C-64 ebenso): In irgendeiner Zeile des Programms soll der Vermerk REM (C) H.CRACKER erscheinen. Bei jedem Versuch, allein diese Zeile zu löschen, soll die Fehlermeldung „Syntax Error" erscheinen, so daß man nur das gesamte Programm löschen kann, was aber auch nicht Sinn der Sache wäre. Diesem Unterfangen kommt eine Eigenart des von Microsoft geschriebenen Basic-Interpreters entgegen, nämlich die, daß die höchste erlaubte Zeilennummer 63999 ist.

Natürlich nimmt die folgende Beschreibung dem Verfahren den Wind aus den Segeln, nachdem aber vergleichsweise wenig Computerbesitzer bis in die tiefsten Innereien ihres Computers hinabsteigen und ferner (zum Leidwesen des Autors)

nur ein paar Tausend dieses Buch besitzen, sei es doch vorgestellt, weil es zumindest bei Nur-Basic-Programmierern Erstaunen hervorruft.

Das kurze Demo-Programm in *Abb. 5.1.3.1* diene als Beispiel. Beim Apple-II werden Basic-Programme ab der Hex-Adresse 0801 abgespeichert. Jede Zeile beginnt mit zwei Bytes, die die Anfangsadresse der nächsten Zeile (nieder- und höherwertiger Teil) enthalten. Sind diese beiden Bytes null, markieren sie das Programmende. Die nächsten beiden Bytes enthalten den nieder- und höherwertigen Teil der Zeilennummer. (Das wurde bereits im Kapitel über Interpreter beschrieben.)

Die Zeile mit dem Copyright-Vermerk hat zunächst noch die Nummer 100. Man kann, wie in der Abbildung gezeigt, nun beim Apple-II mit CALL-151 das Monitorprogramm aufrufen und die Zeile 100 suchen. Dazu muß man wissen, daß die Dezimalzahl 100 hexadezimal 64 ist, intern also (höherwertiger Teil 00 zuletzt) als Bytefolge 64 00 gespeichert wird. (Die Zeilennummer 10 der ersten Zeile taucht als Bytefolge 0A 00 auf.)

Nun kann man die beiden Bytes mit der Zeilennummer 100 z.B. auf FF FF setzen, was der Dezimalzahl 65535 entspricht. Geht man dann mit CTRL-C und Return in den Basic-Interpreter zurück, gibt LIST ein und versucht, die REM-Zeile 65535 zu löschen, so erhält man lakonisch die Meldung „Syntax Error".

Übrigens kann man mit diesem Verfahren auch die Reihenfolge von Zeilennummern im Programm ändern oder sogar mehreren Zeilen die gleiche Nummer geben. Vorsicht ist dann lediglich bei der Verwendung von GOTO- und GOSUB-Befehlen gegeben, da diese die Suche nach der Zieladresse abbrechen, sobald eine größere als die gesuchte Zeilennummer auftritt, und ebenfalls Nummern über 63999 nicht akzeptieren.

Ein weiteres Verfahren, um Programmänderungen zu verhindern, ist, den Namen des jeweiligen Programms erst gar nicht im Disk-Inhaltsverzeichnis erscheinen zu lassen. Im Kapitel über Apple-DOS wurde bereits eine Möglichkeit vorgestellt: Besteht der Name z.B. nur aus dem Buchstaben A, so läßt man ihm noch ein Cursor-nach-links-Steuerzeichen (Backspace) und einen Leerraum folgen, so daß er sich auf dem Bildschirm bei der Anzeige des Inhaltsverzeichnisses sozusagen selbst überschreibt. Um z.B. ein Maschinenprogramm im Adressenbereich hex 0300...03CF so abzuspeichern, gibt man beim Apple-II ein:

```
10 PRINT CHR$(4)+"BSAVE A"+CHR$(8)+"  "+",A$300,L$D0"
RUN
```

Natürlich muß man aber dieses Programm auch irgendwie starten können. Dazu schreibt man sich ein kurzes Startprogramm, das unter irgendeinem Namen (z.B. START) auf Disk abgespeichert wird und mit RUN START aufgerufen werden kann. Es braucht nur eine einzige Zeile zu enthalten. Zunächst sollte es etwa so aussehen:

```
10 PRINT CHR$(4)+"BRUN A  "
```

```
10PRINTCHR$(4)+"BRUN A   "

>CALL-151

*800.817

0800- 00 16 08 0A 00 BA E7 28
0808- 34 29 C8 22 42 52 55 4E
0810- 20 41 20 20 22 00 00 00
*812:08
```

Abb. 5.1.3.2
So wird ein „unsichtbarer" Dateiname erzeugt. Der erste der beiden Leerräume nach dem A wird von Hand mit dem Backspace-Code 08 überschrieben

Wichtig ist, daß nach dem A genau zwei Leerräume folgen. Der erste muß nun durch ein Backspace-Zeichen ersetzt werden. Dazu muß man wieder mit CALL −151 in den Monitor gehen, ab Adresse 0800 nachsehen, wo das A gespeichert ist (ASCII hex 41) und den darauf folgenden ersten Leerraum (hex 20) durch ein Backspace-Zeichen (hex 08) ersetzen. *Abb. 5.1.3.2* zeigt das Vorgehen detailliert. Wenn man dann wieder mit CTRL-C und Return in Basic geht und das Programm listet, stellt man fest, daß der Dateiname des aufzurufenden Programms nun wie im Disk-Inhaltsverzeichnis unsichtbar geworden ist.

Natürlich gibt es noch viel mehr solche Tricks, für jeden Rechnertyp andere. Zuviel soll aber nicht verraten werden, denn Tricks, die jeder kennt, sind wirkungslos ...

Kopierschutz

Am wirksamsten ist es natürlich, wenn die ganze Diskette (Kassetten spielen heute eine untergeordnete Rolle) vor dem Kopieren geschützt ist. Der Schlüssel hierzu liegt normalerweise im Disk-Betriebssystem (DOS), das beim „Booten" der Diskette automatisch in den Speicher geladen wird und alle weiteren Disk-Zugriffe organisiert.

Um zu verhindern, daß mit einem (herkömmlichen) Kopierprogramm die Diskette komplett, also Spur für Spur und Sektor für Sektor, kopiert werden kann, muß die Art der Aufzeichnung selbst verändert werden. Bei Computern wie dem Apple-II, die mit der GCR-Methode arbeiten (Group Code Recording, siehe Kapitel über Floppy-Disk-Aufzeichnung) kann man das zum Beispiel erreichen, indem man einige der 8-Bit-Codes, die 6-Bit-Gruppen zugeordnet sind, vertauscht. Beim Apple-II stehen die zugehörigen Pre- und Post-Nibbelizing-Tabellen ab der Adresse BA29 (48-KByte-System). Ebenso könnte man den Algorithmus zur Berechnung der Sektor-Prüfsumme im DOS ändern, so daß ein Kopierprogramm sofort mit einer Fehlermeldung abbrechen würde.

Voraussetzung dafür, daß solche Methoden greifen, ist natürlich, daß man nicht z.B. durch einfachen Druck auf die Break- oder Reset-Taste das zu schützende Programm (wenn es erst einmal geladen ist) abbrechen und dann mit Hilfe der eventuell im ROM vorhandenen Routinen mit SAVE auf eine Tonbandkassette retten kann, wenn das geladene DOS nicht wie sonst funktioniert. Die Break-Taste

abzufangen, ist selten ein Problem; das Abfangen eines Reset ist dagegen u.U. sogar unnötig: So besitzt z.B. der Commodore-CBM erstens ohnehin keine Reset-Taste, andererseits würde der Gebrauch einer nachträglich eingebauten Taste jedesmal den gesamten Arbeitsspeicher mit hex AA überschreiben (damit prüft der Basic-Interpreter den freien Speicherplatz).

Bei anderen Rechnern, z.B. beim Apple-II, gibt es dagegen einen Reset-Vektor im RAM. Das Monitorprogramm unterscheidet bei einem Reset an Hand einer zusätzlichen Speicherzelle jedesmal, ob es sich um einen Kalt- oder Warmstart handelt. Im ersten Fall wird u.a. der bewußte Reset-Vektor im RAM zunächst umgangen und mit der normalen Reset-Adresse belegt, andernfalls erfolgt ein Sprung über den Reset-Vektor.

Beim Apple-II steht dieser Reset-Vektor im RAM an den Adressen 03F2 und 03F3. Enthält die Speicherzelle 03F4 (Power-Up-Byte) das Ergebnis einer Exklusiv-Oder-Verknüpfung der Bytes in 03F2 und 03F3, so wird ein „warmer" Reset über den RAM-Vektor durchgeführt, andernfalls erfolgt ein Kaltstart zum Neubooten der Diskette. Beispiel: Normalerweise steht bei 03F2 die Bytefolge A5 E0, die der Reset-Adresse E0A5 entspricht. A5 und E0 ergeben, exklusiv-Oder-verknüpft, das Power-Up-Byte hex 45. Nur wenn dieser Wert in der Zelle 03F4 steht, erfolgt bei einem Reset ein Sprung über den RAM-Vektor.

Ein geändertes DOS auf der geschützten Diskette kann diesen Reset-Vektor und das Power-Up-Byte so setzen, daß bei jedem Druck auf Reset entweder das Programm neu gestartet, der RAM-Inhalt völlig gelöscht oder die Diskette einfach neu gebootet wird.

Eine beliebte Methode ist auch, in einem geänderten DOS die Ansteuer-Routine für den Laufwerkskopf-Schrittmotor zu ändern, so daß sich ein anderer Spurabstand als sonst ergibt. Die Durchführbarkeit dieses Kopierschutzes hängt natürlich vor allem von den Gegebenheiten der Laufwerks-Mechanik ab. Insbesondere kann sie dazu führen, daß so geschützte Disketten von bestimmten Laufwerksfabrikaten, die bei „normaler" Software bei den üblichen Spurabständen einwandfrei funktionieren, nicht mehr geladen werden können.

Eine Methode, die recht sicher das Duplizieren kompletter Disketten (allerdings für sich allein noch nicht das Kopieren einzelner Dateien darauf) verhindert, ist, bestimmte Spuren bzw. Sektoren, die nicht von Programmen und Daten belegt sind, gewaltsam zu beschädigen bzw. schon beim Formatieren auszulassen. Wenn ein normales Kopierprogramm auf einen solchen Sektor stößt, bricht es seine Tätigkeit sofort mit einer Fehlermeldung ab.

Allerdings gibt es insbesondere für Computer wie IBM-PC und Apple-II inzwischen eine Vielzahl von Spezialkopierprogrammen, die nicht nur unterschiedliche Spurabstände berücksichtigen, sondern Bit für Bit jeden Sektor kopieren, unabhängig davon, ob er nun nach Normal-DOS-Maßstäben sinnvolle Daten enthält oder nicht (Locksmith, Nibbles Away u.a.).

Der beste Kopierschutz ist deshalb nach wie vor, ein Programm zu einem Preis anzubieten, bei dem sich illegales Kopieren einfach nicht lohnt, und die Dokumentation dazu so gut zu machen, daß allein sie den Preis wert ist.

5.1.4 Paßwort-Schutz

Der Schutz von Daten ist in der Bundesrepublik Deutschland und in vielen anderen Ländern gesetzlich geregelt (Datenschutzgesetz). In größeren Firmen gibt es einen eigenen Datenschutz-Beauftragten, der über die Einhaltung der Bestimmungen zu wachen hat. Doch kann er sein Auge nicht überall haben, und am wirksamsten (wenn auch dafür kein Ersatz) sind deshalb doch technische Verfahren, mit denen der Zugang zu bestimmten Rechnern, Programmen und Daten auf den dafür berechtigten Personenkreis begrenzt wird.

Geheime Schlüssel

Wer mit einer magnetisch beschichteten Scheckkarte von einem Bargeld-Automat einige Scheine holt und sich Auskunft über den aktuellen Kontostand geben läßt, benötigt dazu außer der Scheckkarte selbst eine Geheimnummer, die aus mehreren Ziffern besteht und die er, sobald die Karte in den Automat eingeschoben wurde, zusätzlich eintippen muß. Dieses Verfahren haben die Banken gewählt, damit niemand mit einer geraubten, gestohlenen oder gefundenen Karte Unheil auf dem Konto des rechtmäßigen Besitzers anrichten kann. Die Geheimnummer ist nur dem Rechner der Bank und dem Besitzer der Karte bekannt; man erhält sie bei der Bank als Computer-Ausdruck in einem verschlossenen Umschlag zusammen mit der Karte.

Als Benutzer des Bildschirmtext-Dienstes der Deutschen Bundespost muß man nach dem Herstellen der Telefonverbindung zum zentralen Rechner ebenfalls zunächst ein geheimes Paßwort eingeben, das man selbst bei Bedarf auch ändern kann (zum Beispiel, wenn einem am Neujahrsmorgen einfällt, es in angeheitertem Zustand am Silvester-Abend ausgeplaudert zu haben).

Bei Multiuser-Betriebssystemen (z.B. Unix), bei Tischcomputern (z.B. Pronto-16), ja sogar bei Basic-Taschencomputern (Sharp-PC-1XXX-Serie) ist ebenfalls eine Zugriffssperre mit Paßworten möglich, im letzteren Fall zum Sperren eines gespeicherten Programms vor unbefugter Auflistung und Änderung.

All dies sind Beispiele für den Datenschutz mit geheimen Paßworten. Paßworte werden, analog zu Türschlössern, auch als Schlüssel bezeichnet. Für die Geheimhaltung und damit die Aufrechterhaltung des Datenschutzes bzw. der Zugangsbeschränkung zu einem Rechner ist der Träger selbst verantwortlich. Wer das Geburtsdatum der Tochter, die Telefonnummer des Freundes oder so geniale Worte wie „GEHEIM“ als Paßwort verwendet, braucht sich nicht zu wundern, daß er

nicht lange der alleinige Zugriffs-Berechtigte bleibt; wenn das Geld so auf der Straße liegt, warum sollte es ein „Hacker" — so nennt man die Paßwort-Knacker — nicht einfach aufheben ...

Manchmal kommt den so gefürchteten Hackern allerdings auch einfach Kollege Zufall zu Hilfe. Hierzu ein Zahlenbeispiel. Ein Rechenzentrum habe 1000 registrierte, legale Benutzer. Jeder Benutzer erhält eine vierstellige Benutzernummer (0000 bis 9999) zur Unterscheidung von den anderen, ferner eine dreistellige Geheimnummer (000...999), damit er sich beim Rechner als registrierter Teilnehmer ausweisen kann.

Wenn ein Außenstehender die Telefonnummer des Rechners und Art und Länge der Benutzer- und Geheimnummer erfährt (z.B. durch unachtsame Äußerungen eines Bekannten, der selbst legaler Teilnehmer ist), hat er rein statistisch nach nur zehn Anrufen durch Probieren schon eine gültige Benutzernummer herausgefunden, da 1000 von 10000 möglichen Nummern tatsächlich zu Teilnehmern gehören. Eventuell genügen sogar weniger Versuche, wenn die gültigen Nummern nämlich vom Rechenzentrum unmittelbar aufeinanderfolgend vergeben wurden.

Der nächste Schritt betrifft das Herausfinden der Geheimnummer. Hier gibt es für die gerade gefundene Benutzernummer genau 1000 Möglichkeiten. Vom Glück hängt es dabei ab, wie viele Versuche es kostet, bis man es geschafft hat; auch wenn der Rechner bei einer Falscheingabe nicht jedesmal die Telefonleitung kappt, kostet es einige Zeit und Telefongebühren. Aber gerade die Statistik sagt, daß es mit etwas Glück auch gleich beim ersten Mal klappen kann. Genau dieser Fall kommt statistisch bei jedem tausendsten Hacker, der es probiert, einmal vor.

Folgende Regeln sind also bei Verwendung von geheimen Paßworten unbedingt zu beachten:

1. Das Paßwort darf nicht zu kurz sein, da man es sonst durch bloßes Probieren zu schnell findet, und muß unter allen Umständen geheim bleiben (also kein Zettel unterm Schreibtisch u.ä.!).
2. Aus persönlichen Daten wie Geburtstag usw. darf kein Rückschluß auf das Paßwort möglich sein.
3. Auch Teilinformationen wie Telefonanschluß des Rechners, Aufbau von Benutzernummern usw. dürfen nicht an Dritte gegeben werden.
4. Falls es Anzeichen dafür gibt, daß noch jemand das Paßwort kennt, sollte man es sofort ändern und ggf. auch dem Betreiber des Rechners Mitteilung machen (möglicherweise wurden Daten geändert, ohne daß dies sofort auffällt).

Öffentliche Paßworte

Bei sehr wertvollen Daten muß man bedenken, daß es Leute geben könnte, die auch sehr viel Mühe und Zeit investieren, an diese heranzukommen. Das kann bis zum Anzapfen von Telefonleitungen gehen, um Modem-Datenübertragungen mit-

zuschneiden und dadurch auch das verwendete Zugriffs-Paßwort zu decodieren. In solchen Fällen ist ein Paßwort, das immer gleich bleibt oder dessen Änderung durch den Benutzer ebenfalls von Dritten abgehört werden könnte, ungeeignet. Eine Alternative stellen sogenannte „Public Keys" dar (öffentliche Schlüssel).

Public Keys basieren auf folgendem Prinzip: Der angerufene Rechner sendet zunächst eine Zufalls-Zeichenfolge (z.B. eine vierstellige Zufallszahl). Der anrufende Benutzer muß diese nun entweder im Kopf oder mit Hilfe eines Taschenrechners oder gar eines Tischcomputers mit Hilfe eines allein ihm bekannten Algorithmus (Rechenvorschrift, z.B. eine Formel) in eine Ergebniszahl umrechnen, die er an den angerufenen Rechner zurücksendet. Da inzwischen der Rechner auf die vorher ausgesendete Zufallszahl den gleichen Algorithmus angewendet hat, braucht er nur sein eigenes Ergebnis mit dem des Anrufers zu vergleichen und kann daran sofort sehen, ob es sich um einen legalen Zugang handelt.

Der verwendete Algorithmus muß einigen Forderungen genügen. So sollte er zum Beispiel den zulässigen Wertebereich (Zahlenbereich) möglichst vollständig und gleichmäßig ausschöpfen: Die erwarteten Ergebnisse sollten sich z.B. nicht im Bereich 450...550 häufen, wenn vierstellige Zahlen zulässig sind. Ferner sollte es sich nicht um eine regelmäßige oder lineare Funktion handeln, auf deren Verlauf man schon bei Kenntnis einiger weniger Werte Rückschlüsse ziehen kann, wie etwa Y=32*X−17. Optimal ist in dieser Hinsicht eine Tabelle aus Zufallswerten, für die die vom Rechner ausgesendete Zahl (selbst schon zufällig) als Index dient. Aber auch komplexe Polynome folgender Form sind geeignet:

$$Y = A1*X + A2*X\uparrow 2 + A3*X\uparrow 3 + A4*X\uparrow 4 \ldots + An*X\uparrow n$$

X ist dabei die vom Rechner gesendete Zufallszahl, A1...An beliebig vereinbarte Koeffizienten (z.B. −3.1, +5.9 usw.) und Y die zurückzusendende Ergebniszahl. Die Koeffizienten sind so zu wählen, daß die Ergebnisse im beabsichtigten Wertebereich (z.B. 0...9999) liegen.

Noch einfacher kann man ein solches Transformations-Paßwort, wie man das Verfahren auch nennt, unter Zuhilfenahme der INT-Funktion gewinnen, die den ganzzahligen Anteil einer Zahl extrahiert. Ein recht brauchbares kleines Basic-Programm zum Errechnen eines Transformations-Paßworts wäre nämlich auch zum Beispiel:

```
10 INPUT X
20 Y=(X+43)/13
30 Y=INT(9000*(Y-INT(Y)))+1000
40 PRINT Y
```

Die vom angerufenen Rechner gelieferte Zufallszahl X wird hierbei mit einer unstetig verlaufenden Funktion in ein Ergebnis umgerechnet, das im Wertebereich 1000...9999 liegt. Hierzu wird lediglich der Nachkomma-Anteil der Funktion von Zeile 20 verwendet, der fast zufällige Werte liefert. Der Bereich unter 1000 wurde absichtlich vermieden, um vierstellige Paßwörter sicherzustellen. Das Programm

erfordert einen Interpreter oder Compiler mit wenigstens acht Stellen Genauigkeit. Es läßt sich beispielsweise in einem Basic-Taschencomputer unterbringen und selbst gegen Auflistung sperren (bei den meisten Sharp-Taschencomputern gibt es dazu das Kommando PASS "Paßwort").

Verschlüsseln von Daten

Auch für das Ver- und Entschlüsseln von Daten verwendet man Paßworte. Damit ist es möglich, Daten auf Disketten oder per Telefonmodem zu versenden, die nicht an Dritte geraten dürfen (z.B. aus Gründen des Datenschutzes). Dafür sind sehr unterschiedliche und trickreiche Verfahren in Gebrauch. Grundsätzlich sind an Verschlüsselungs-Methoden folgende Forderungen zu stellen:

1. Die verschlüsselten Daten sollten auch durch statistische Auswertung z.B. der typischen Häufigkeit bestimmter Buchstaben in der Sprache keinen Rückschluß auf die Originaldaten zulassen.
2. Die verschlüsselten Daten sollten nicht wesentlich mehr Speicherplatz einnehmen als die ursprünglichen.
3. Es sollten möglichst viele unterschiedliche Schlüssel (Paßworte) möglich sein.

Grundsätzlich unterscheidet die Kryptografie, wie man die Wissenschaft der Ver- und Entschlüsselung auch nennt, zwei Methoden, die auch beide praktisch Verwendung finden:

a) Vertauschung der Reihenfolge der zu codierenden Zeichen.

b) Umcodieren jedes einzelnen Zeichens bei gleichbleibender Reihenfolge.

Die Methode a ist besonders dort sinnvoll einzusetzen, wo Datenblöcke immer gleichbleibender Länge übertragen werden sollen; gegebenenfalls kann man solche auch durch Aufspalten der Daten bilden, beispielsweise bei der „verwürfelten" Übertragung von Sprache z.B. über Funk: Die vom Mikrofon her kommenden Signale werden einem Analog-Digital-Umsetzer zugeleitet. Die resultierenden Digitalwerte werden in einem Pufferspeicher in ihrer Reihenfolge nach einem festgelegten Algorithmus (Pseudo-Zufallszahlen-Folge) „verwürfelt". Auf der Empfängerseite wird die ursprüngliche Reihenfolge wieder hergestellt, über einen D/A-Wandler erhält man verständliche Sprache zurück.

Ein beliebtes, meist verwendetes Verfahren der Verschlüsselung nach Methode b ist die zyklische, bitweise Exklusiv-Oder-Verknüpfung jedes Bytes mit einem Paßwort. Diese Methode hat zugleich den Vorteil, daß das hierfür nötige Programm zur Ver- und Entschlüsselung geeignet ist. Ein dafür geeignetes 6502-Programm, mit dem Assembler „Lisa" auf einem Apple-II programmiert, zeigt *Abb. 5.1.4.1.* Es holt sich Anfangsadresse und Länge des zu ver- oder entschlüsselnden Bereichs aus den DOS-Adressen AA72/AA73 und AA60/AA61 (48-KByte-Apple). Dort werden diese Parameter nämlich nach einem BLOAD-Kommando automatisch richtig abgelegt (wenn also das abgebildete Programm schon ab hex 0300 im Speicher steht und das zu verschlüsselnde mit BLOAD von Disk geladen wird).

```
0800             1  ;VER- UND ENTSCHLUESSELUNG
0800             2  ;DURCH EXKLUSIV-ODER-FOLGE
0800             3  START   EQU $AA72            ;STARTADRESSE
0800             4  LEN     EQU $AA60            ;LAENGE
0800             5  PNT     EPZ $FC              ;ADR.-ZEIGER
0800             6  CNT     EPZ $FE              ;BYTEZAEHLER
0300             7          ORG $300
0300 AD72AA      8  KEY     LDA START            ;STARTADR.
0303 85FC        9          STA PNT              ;LADEN
0305 AD73AA     10          LDA START+1
0308 85FD       11          STA PNT+1
030A A900       12          LDA #0               ;ZAEHLER
030C 85FE       13          STA CNT              ;RUECKSTELLEN
030E 85FF       14          STA CNT+1
0310 A200       15  KEY0    LDX #0               ;INDEX AUF
0312 A000       16          LDY #0               ;SCHLUESSEL
0314 BD3B03     17  KEY1    LDA WORD,X
0317 F0F7       18          BEQ KEY0             ;WORD-ENDE
0319 51FC       19          EOR (PNT),Y          ;EXKL.-ODER
031B 91FC       20          STA (PNT),Y
031D E6FC       21          INC PNT              ;ZEIGER
031F D002       22          BNE KEY2             ;WEITER-
0321 E6FD       23          INC PNT+1            ;STELLEN
0323 E6FE       24  KEY2    INC CNT              ;ZAEHLER
0325 D002       25          BNE KEY3             ;INKREM.
0327 E6FF       26          INC CNT+1
0329 A5FE       27  KEY3    LDA CNT              ;AUF ENDE
032B CD60AA     28          CMP LEN              ;PRUEFEN
032E D007       29          BNE KEY4
0330 AD61AA     30          LDA LEN+1
0333 C5FF       31          CMP CNT+1
0335 F003       32          BEQ RDY
0337 E8         33  KEY4    INX                  ;NAECHSTES
0338 D0DA       34          BNE KEY1             ;BYTE
033A 60         35  RDY     RTS                  ;FERTIG
033B FE0519     36  WORD    HEX FE05198400
033E 8400
                37          END
```

Abb. 5.1.4.1 Programm zur Exklusiv-Oder-Verschlüsselung eines Speicherbereichs mit der in Zeile 36 stehenden Bytefolge, mit dem Apple-Assembler Lisa geschrieben

Ab Adresse 033B steht der Schlüssel. Er kann bis 255 Bytes lang sein (je länger, desto sicherer); sein Ende ist mit einem Null-Byte markiert. Das erste Schlüsselbyte wird auf das erste Byte des zu codierenden Speicherbereichs angewendet, das zweite auf das zweite und so weiter. Nach dem letzten Schlüsselbyte wird wieder mit dem ersten angefangen, bis der gesamte Speicherbereich codiert ist. Wie das in der Praxis aussieht, macht *Abb. 5.1.4.2* deutlich: Im Bereich 0800 bis 080F steht zur Demonstration die Bytefolge 01, 02, 03 usw. bis 0F. Ihre Anfangsadresse wird in AA72/AA73 geschrieben (00; 08; niederwertiger Teil zuerst), die Länge von hex 10 Bytes in AA60/AA61 (10; 00).

```
*800.80F

0800- 00 01 02 03 04 05 06 07
0808- 08 09 0A 0B 0C 0D 0E 0F
*AA72:00 08

*AA60:10 00

*300G

*800.80F

0800- FE 04 1B 87 FA 00 1F 83
0808- F6 0C 13 8F F2 08 17 8B
*300G

*800.80F

0800- 00 01 02 03 04 05 06 07
0808- 08 09 0A 0B 0C 0D 0E 0F
```

Abb. 5.1.4.2
Ver- und Entschlüsselung des Speicherbereichs 0800 bis 080F mit Hilfe des Programms aus Abb. 5.1.4.1

Mit dem Monitor-Kommando 300 G wird das Verschlüsselungs-Programm gestartet; der Speicherbereich 0800...080F ist nun kaum wiederzuerkennen. Ein nochmaliges 300 G entschlüsselt die Bytefolge wieder.

Wie entschlüsselt man nun derart verschlüsselte Daten, wenn man den Schlüssel nicht kennt oder vergessen hat? Wenn die ursprünglichen Daten viele aufeinanderfolgende Null-Bytes enthielten (wesentlich mehr, als der Schlüssel lang ist), so kann man an diesen Stellen der codierten Daten sehen, wie sich eine bestimmte Bytefolge wiederholt: Sie ist unmittelbar der Schlüssel selbst. Wenn es sich nicht um Null-Bytes handelte, dann sind es vielleicht Leerräume im Text; sie besitzen den Code hex 20. Den Schlüssel erhält man dann, indem man jedes Byte der sich sichtbar wiederholenden Folge mit hex 20 exklusiv-oder-verknüpft.

Selbstverständlich sind auch Kombinationen der beiden oben genannten Methoden a und b (Vertauschen der Reihenfolge und gleichzeitiges Umcodieren jedes einzelnen Zeichens) möglich, was zu sehr effizienten und sicheren Verschlüsselungs-Methoden führt.

5.2 Entwicklung, Debugging, Marketing

5.2.1 Entwicklungsphasen

Wer als Hobby-Programmierer ab und zu ein Programm schreibt, braucht sich nicht um Effizienz der Arbeit, Einhaltung von Terminen oder Vermarktbarkeit zu

kümmern. Im Bereich kommerzieller Software-Entwicklung wird ein Projekt dagegen oft gemeinsam von mehreren Fachleuten „durchgezogen", was eine exakte Koordination erfordert. Übrigens ist es unbedingt sinnvoll, jeden einzelnen Abschnitt der Entwicklung sorgfältig zu dokumentieren, sonst hat man irgendwann vergessen, was genau man mit einer bestimmten Anweisungsfolge eigentlich wollte.

Die hier empfohlene Vorgehensweise ist im „Software Engineering" (die Wissenschaft vom Programmschreiben) als „Phasenkonzept" geläufig. Es ist sehr wichtig, daß dabei weder die Reihenfolge der einzelnen Entstehungsphasen vertauscht wird (also nicht erst Schreiben einer Detailroutine, dann Festlegung, wie man sie im Gesamtprogramm sinnvoll nutzen kann) noch zwischen den Phasen hin- und hergesprungen wird (nachträgliche Korrekturen des Gesamtkonzepts, wenn man merkt, daß ein Detailproblem nicht ganz einfach zu lösen ist).

Vorüberlegungen

Ein größeres Entwicklungsprojekt verlangt eine sorgfältige Planung, um dem Kunden bzw. der Marketing-Abteilung den voraussichtlichen Fertigstellungs-Termin und die zu erwartenden Kosten im voraus mitteilen zu können. Unter Umständen ergibt diese Abschätzung nämlich schon im Vorfeld, daß sich die ganze Entwicklung eigentlich nicht lohnen würde, weil z.B. kein marktgerechter Preis erzielt werden kann oder der vom Kunden verlangte Termin nicht realistisch ist.

Möglicherweise stellt sich bei den Vorüberlegungen auch heraus, daß es schon Software gibt (entweder aus dem eigenen Hause oder aus fremden Quellen), die nach geringfügiger Anpassung das Problem bzw. einen Teil davon löst und so die Entwicklungszeit verkürzt.

Erstellen eines Grob-Konzepts

Hat sich ergeben, daß sich die Entwicklung lohnt, muß man nun prinzipielle Lösungswege (Konzepte) suchen und vergleichen (Brainstorming). Bei der Entwicklung eines Texteditors könnte das zum Beispiel sein, ob dieser bildschirm- oder zeilenorientiert arbeitet und ob der gesamte Text im RAM bearbeitet wird oder ob immer nur ein Teil automatisch von Disk geladen wird.

Hier sollte man noch nicht zu sehr ins Detail gehen, da man sich andernfalls elegante Lösungen in späteren Stadien verbauen könnte. Andererseits sollte man sich genau überlegen, für welches der möglichen Konzepte man sich letztendlich entscheidet, denn ein Fehler in dieser Phase kann schwerwiegende Folgen haben. Insbesondere muß man immer im Auge behalten, welche Möglichkeit bei der vorgegebenen Hardware (z.B. 8-Bit-Rechner mit 32 KByte RAM) überhaupt realistisch ist.

Definition der Eigenschaften

Erst jetzt sollte man daran gehen, das Verhalten und die Wirkungsweise des Programms exakt zu definieren. In unserem Beispiel, dem Texteditor, gehören dazu

die Festlegung der Benutzer-Oberfläche (Kommandos, Bildschirm-Aufbau usw.) sowie die Festlegung einzelner Programm-Moduln, die voneinander unabhängig entwickelt und getestet werden können und die Aufgabe in überschaubare Einzelprobleme aufteilen, zum Beispiel Text- und Kommando-Eingabe, Programmkern und Ausgabe von formatiertem Text und eventuellen Systemmeldungen.

Theoretisch könnte man weite Teile der Bedienungsanleitung zum Programm schon jetzt schreiben. Wichtige Kriterien, die schon in dieser Phase berücksichtigt werden müssen, sind:

1. Erfüllung der definierten Anforderungen (Rechengenauigkeit, verarbeitbare Datenmengen usw.).
2. Zuverlässigkeit: Auch unter ungewöhnlichen Randbedingungen muß die Funktion gewährleistet sein (z.B. Abfangen von Fehlermeldungen bei Division durch Null u.a.).
3. Sicherheit: Sinnvolle Reaktion auf unlogische Eingaben, zusätzliche Abfragen bei „kritischen" Kommandos (z.B. zum Löschen von Dateien, Paßwort-Schutz).
4. Benutzerfreundlichkeit (Ergonomie): Leicht merkbare Kommandos, sinnvoller Bildschirm-Aufbau, Menüs usw.
5. Effizienz: Speicherplatzbedarf und Geschwindigkeit müssen in einer vernünftigen Relation zur Leistung des Programms stehen.

Entwurf des Programms

Nachdem man sich darüber im klaren ist, aus welchen Moduln die Software bestehen soll, kann man darangehen, deren Aufgaben im Detail festzulegen. Jedes Modul sollte über nur einen Eingang und auch nur über einen Ausgang verfügen (normalerweise ist der Eintrittspunkt am Anfang und der Ausgang z.B. in Form eines RETURN-Befehls am Ende eines Abschnitts).

Zum Entwurf des Programms gehört auch die Entscheidung, in welcher Programmiersprache es zu schreiben ist. Eventuell kommt auch eine Kombination in Betracht, z.B. ein Basic-Rahmenprogramm und ein in Assembler geschriebener Teil für zeitkritische und maschinennahe Aufgaben.

Als Entwurfshilfsmittel kann man zum Beispiel Flußdiagramme oder Struktogramme einsetzen. Jedes einzelne Programm-Modul sollte dabei klein genug sein, daß es sich auf etwa einer DIN-A4-Seite vollständig beschreiben läßt.

Codierung der Moduln

Schließlich wird jedes Modul einzeln in der dafür festgelegten Programmiersprache geschrieben. Dabei ist auf wesentliche weitere Kriterien zu achten:

1. Möglichkeit späterer Änderungen. Es gibt kein Programm ohne jegliche Fehler, also muß es so dokumentiert sein, daß Änderungen eventuell auch von einem anderen Programmierer ohne weiteres später durchgeführt werden können.
2. Prüfbarkeit. Wenn man sich an den Grundsatz hält, daß jedes Modul nur einen Eingang und nur einen Ausgang besitzt, kann man es unabhängig von anderen durch Parameter-Simulation testen und so Fehler frühzeitig korrigieren.
3. Portabilität, also einfache Übertragbarkeit auf andere Rechnertypen. Das ist insbesondere durch Verwendung höherer Programmiersprachen gewährleistbar.
4. Spätere Verwendbarkeit. Die einzelnen Moduln sollten möglichst so geschrieben sein, daß sie nicht zu eng an das aktuelle Problem gebunden sind, sondern sich bei späteren Programmieraufträgen leicht weiterverwenden lassen.

Der letzte Punkt läuft auf sogenannte Bibliotheks-Moduln hinaus. Unter Umständen werden solche Moduln auch von Kollegen, die an anderen Teilproblemen arbeiten, gelegentlich mitbenutzt. Man sollte daher sehr vorsichtig sein, sie ohne Vorwarnung zu ändern, da dies für die übrigen Programmierer schlimme Folgen haben kann.

Modultest

Wie schon erwähnt, ist ein Test der einzelnen Moduln völlig getrennt voneinander möglich, wenn sie mit genau definierten Übergabe-Parametern arbeiten und auch nur eine Aussprungstelle besitzen.

Indem man systematisch alle Möglichkeiten der unterschiedlichen Eingangs-Parameter durchspielt (auch ungewöhnliche Kombinationen, z.B. der Aufruf einer Routine zur Textspeicherung auf Floppy mit gültigem Namen, obwohl gar kein Text im Speicher steht), kann man theoretisch alle denkbaren noch vorhandenen Fehler dieses Programmteils finden.

Würde man den Test nicht getrennt an den einzelnen Moduln, sondern erst am Gesamtprogramm durchführen, so wäre wegen der Vielzahl der dann denkbaren Parameter-Kombinationen eine vollständige Prüfung weit schwieriger, oft sogar praktisch unmöglich.

Integration

Bei der Software-Entwicklung versteht man unter „Integration" nicht etwa das Herstellen integrierter Schaltungen, sondern das Zusammenfügen der einzeln entwickelten Moduln zu einem Gesamtprogramm. Wenn der Test der Moduln vorher

nicht schlampig durchgeführt wurde, dürfen sich hierbei eigentlich kaum noch Probleme ergeben, es sei denn, das Gesamtprogramm ist nun plötzlich zu lang und läßt im Arbeitsspeicher nicht mehr genug Platz für Daten (z.B. für den zu bearbeitenden Text).

Nachdem der Kern des Bedienungs-Handbuchs zum Programm schon in der Definitions-Phase entstanden ist, kann es nun mit wenig Aufwand vervollständigt werden. Es ist wichtig, daß es beim nachfolgenden Gesamt-Test zumindest als Rohmanuskript zur Verfügung steht, da seine Verständlichkeit wesentlich zur Benutzerfreundlichkeit des nun entstandenen Software-Produkts beiträgt.

Test des Software-Pakets

Bevor man ein umfangreiches Programm verkauft, ist es unbedingt notwendig, es von Personen prüfen zu lassen, die der angepeilten Zielgruppe entsprechen (bei einem Texteditor zum Beispiel eine Büro-Schreibkraft). Software-Entwickler werden von Nicht-Computerfreaks als ganz sonderbare Leute mit viereckigen Augen (wegen der dauernden Bildschirmarbeit ...) und leicht rauchenden Köpfen angesehen, und tatsächlich hat ein Programmierer, der die Position der vier Cursor-Tasten auf dem Keyboard schon auswendig weiß, überhaupt kein Vorstellungsvermögen, was für Tasten eine nicht vorgebildete Büro-Schreibkraft alle durchprobiert, nur um den zuletzt eingegebenen Buchstaben zu löschen.

Der Test der Software ist unter realistischen Bedingungen nur sinnvoll, wenn er mit der (zunächst) endgültigen Handbuch-Version erfolgen kann. Meist ergeben sich aber gerade für das Handbuch noch wichtige Ergänzungen und Korrekturen, um Verständnis-Schwierigkeiten auszuräumen.

5.2.2 Marketing

Wenn es sich bei dem entwickelten Programm nicht um eine Auftragsarbeit handelt, die ausschließlich an den Auftraggeber verkauft wird, muß man jetzt auf die Suche nach potentiellen Abnehmern gehen. Im Gegensatz zur Groß-EDV hat man es in der Mikrocomputer-Branche nicht mit einigen wenigen Kunden zu tun, die einige tausend Mark für ein Programm investieren können, sondern mit einer Vielzahl von Kunden, die an Software-Preise von 20 DM aufwärts bis etwa zum Anschaffungspreis der Rechner-Hardware gewöhnt sind. Es ist daher hier nicht rentabel, jedem Interessenten einen Berater ins Haus zu schicken.

Mikrocomputer-Software wird im wesentlichen durch die gleichen Vertriebskanäle wie die Hardware verkauft, wobei jeder spezifische Vor- und Nachteile hat:

Vertriebsweg	Vorteile	Nachteile
Regionale Fachhändler	Demonstration vor den Augen der Kunden, Beratung vor Ort, schnelle Problembeseitigung	Höherer Preis wegen Händler-Rabatt erforderlich, Kosten für Händler-Schulungen, große „tote" Stückzahlen für Händler-Lager
Cash-and-Carry-Läden	Relativ geringe Handelsspanne, keine Schulungskosten	Keine Kundenberatung, führt wegen niedrigerer Verkaufspreise zu Zurückhaltung (Preiskrieg) bei den Fachhändlern am Ort, große „tote" Stückzahlen für Lagerhaltung
Direkt-Versand	Konkurrenzfähige Preisgestaltung möglich, minimale Vorfinanzierung wegen geringer Lagerstückzahlen	Vorführung des Programms höchstens auf Messen und Ausstellungen möglich, Kundenbetreuung nur schriftlich und telefonisch

Für den Vertrieb eines weitgehend selbsterklärenden Massenprodukts an Einzelabnehmer kommen am ehesten Cash-and-Carry-Shops in Betracht, für beratungsintensive Programme dagegen der Fachhandel (natürlich muß die Preisgestaltung eine Beratung für den Händler auch lohnend machen). Programme für Spezialanwendungen, bei denen die projektierte Verkaufsstückzahl nur einige Hundert beträgt, sind dagegen mehr für den Direktvertrieb geeignet, da allein schon die Ausstattung aller regionalen Händler mit Programmen, die sie womöglich dann nicht verkaufen, unrentabel wäre.

Wirb oder stirb

Ein Produkt kann noch so gut sein — solange es niemand kennt, kauft es niemand. (Böse Zungen sagen auch: Ein Produkt kann noch so schlecht sein — wenn es jeder kennt, wird's auch jemand kaufen ...) „Wirb oder stirb" ist ein alter und nichtsdestotrotz nach wie vor gültiger Spruch von Werbe-Strategen.

Wie macht man ein Software-Produkt bekannt? Im wesentlichen gibt es dazu folgende Möglichkeiten:

1. Direktversand von Prospekten. Die nötigen Adressen nimmt man entweder aus der Kundenkartei (oft sehr effizient) oder kauft sie z.B. von der Post (!) für einige Pfennige je Name einer Berufsgruppe (Streuversand, selten effizient).
2. Vorstellung auf Messen und Ausstellungen. Leider nur ein Tropfen auf den heißen Stein; Händler kann man so zwar gut kennenlernen, die nötige Zahl an Endkunden jedoch kaum.
3. Werbung in Computer-Fachzeitschriften. Da die meisten engagierten Computer-Anwender mindestens eine Fachzeitschrift (z.B. mc) lesen, ist der Streuver-

lust bei systemnahen Programmen (Betriebssystem-Erweiterungen, Utilities usw.) gering, die Investition von 2000...6000 DM pro A4-Seite lohnt sich.

4. Werbung in sogenannten Special-Interest-Zeitschriften (z.B. PM, Geo usw.), die sich zwar nicht direkt und in erster Linie an Computerbesitzer wenden, die aber ein breiteres Feld ansprechen. Wegen des hohen Streuverlusts ist dies meist wenig effizient.
5. Werbung in berufsspezifischen Zeitschriften (z.B. für Ärzte, Börsen-Spekulanten usw.). Bei Branchen, in denen sich Tischcomputer schon gut durchgesetzt haben, recht effizient.

Pressemitteilungen

Eine weitere Möglichkeit, für Publicity zu sorgen, ist das Versenden einer Pressemitteilung an jene Zeitschriften, in deren Themenspektrum das Produkt paßt. Findet die Redaktion diese Mitteilung interessant, so wird sie sie eventuell in einer der nächsten Ausgaben abdrucken.

Eine solche Pressemitteilung sollte den Fachzeitschriften, die eine wesentlich längere Vorlaufzeit besitzen als z.B. Tages- oder Wochenzeitungen, etwa zwei Monate vor der offiziellen Markteinführung vorliegen. Liegt sie erst vor, wenn der Redakteur das Produkt längst z.B. in Anzeigen oder dem Fachhandel gesehen hat, wird er sie vermutlich als wenig aktuell einstufen, da sie ja frühestens erst zwei Monate später erscheinen kann. Wird sie dagegen schon abgedruckt, bevor die Leser das Produkt wirklich kaufen können, hat man sich schnell den Ruf eingehandelt, nicht liefern zu können.

Eine wirkliche Unsitte ist es bei manchen Firmen, die gleichzeitige Aufgabe von Inseraten (und seien sie noch so winzig) davon abhängig zu machen, daß die Redaktion der gleichen Zeitschrift die übersandte Pressemitteilung möglichst im gleichen Heft berücksichtigt. Jede Redaktion versucht das Heft so zu gestalten, daß eine interessante Mischung herauskommt, die für möglichst alle Leser etwas bietet. Der Versuch von Inserenten, hierauf Einfluß zu nehmen, führt in den meisten Fällen eher zu bösem Blut als zu einer objektiven, glaubwürdigen Berichterstattung, die schon deshalb auch für den Inserenten wichtig ist, weil ohne sie die gerade für den Erfolg seiner Anzeige so wichtige Leserzahl nie erreicht werden könnte.

Eine ebensolche Unsitte ist es, regelmäßig in einer Redaktion nachzufragen, wann eine bestimmte Produktmeldung nun endlich abgedruckt wird. Oft läßt sich der Weg einzelner Produktmeldungen auf ihrem Weg vom Posteingang zum fertigen Heft nicht genau rekonstruieren, weil die Meldungen von unterschiedlichen Redakteuren bearbeitet werden und zum Teil auch je nach Platzverhältnissen vom Hersteller (das ist derjenige im Verlag, der die gesetzten Textfahnen zu Seiten zusammenklebt) selbständig auf das nächste Heft verschoben werden müssen.

Pressemitteilungen sollten, mit doppeltem Zeilenabstand geschrieben, etwa eine, maximal aber zwei Schreibmaschinenseiten umfassen. Zusätzliche ausführlichere Informationen können beigelegt werden, auf jeden Fall sollte eine Kontakt-

adresse für technische Fragen angegeben sein (also nicht nur die Adresse der PR-Agentur, die die Pressemitteilung herausgegeben hat). Hilfreich ist auch ein illustrierendes Foto (schwarzweiß oder in Farbe als wirklich scharfe Hochglanz-Papierbilder ohne Grauschleier oder sichtbares Korn mit gut ausgeleuchteten Objekten, Format mindestens 9 × 13, eventuell auch ein Dia).

Der Text sollte sachlich, nur mit nachprüfbaren Argumenten und nicht unbedingt im Stil von Waschmittel-Werbung geschrieben sein, auch wenn das Produkt noch so gut ist. Folgende Informationen sollte er unbedingt enthalten:

1. Welche Zielgruppe wird genau angesprochen? Bei Software: Welche Hardware-Voraussetzungen müssen erfüllt sein?
2. Was ist so neu und einmalig an diesem Produkt?
3. Wie kann man (am besten tabellarisch) seine Eigenschaften zusammenfassen?
4. Wann und wo wird es zu welchem Preis erhältlich sein?
5. Kann die Redaktion auf Anforderung leihweise ein Muster zu Testzwecken erhalten (wenn ja, wann)?

Bei der Formulierung sollte man übrigens noch rasch einen Blick in den letzten Abschnitt des Kapitels 5.1.1 werfen, um wenigstens die allertypischsten Jargon-Fehler zu vermeiden.

Lizenzvergabe

Für den Verkauf eines Software-Produkts über Händler sind unterschiedliche Verfahren üblich, von denen jedes spezifische Vor- und Nachteile besitzt.

a) Wie bei Hardware auch bekommt der Händler Handbuch nebst Diskette gemeinsam verpackt. Der Händler hat nicht das Recht, Kopien der Software anzufertigen.

b) Der Händler erhält das Recht, Kopien des Programms anzufertigen, bezieht jedoch die Handbücher dazu weiterhin vom Software-Hersteller. Letzterer hat dadurch eine Kontrollmöglichkeit der verkauften Stückzahl.

c) Der Händler erhält das Recht, Kopien des Programms anzufertigen und auch das zugehörige Handbuch selbst zu drucken. Der Software-Hersteller kontrolliert den Verkauf, indem er jedem Händler eine bestimmte Folge von Seriennummern reserviert, die dieser selbst auf die Diskette kopieren muß. Wie bei a und b erfolgt eine Abrechnung nach verkaufter Stückzahl.

d) Der Händler kopiert wie bei c Disketten und Handbücher selbst, nachdem er dem Software-Entwickler eine einmalige Lizenzgebühr bezahlt hat. Eine Überwachung der verkauften Stückzahl findet also nicht statt.

Der organisatorische Aufwand und das mit Vorproduktion und Lagerhaltung verbundene Risiko auf seiten des Software-Herstellers nehmen von a bis d stetig ab, während umgekehrt Aufwand und Risiko beim Händler unter Punkt a am geringsten sind. Auf der anderen Seite kann der Händler gerade mit Punkt d ein Rie-

sengeschäft machen, wenn der Software-Hersteller die Verkaufschancen seines Programms anfänglich zu schlecht eingeschätzt hat und eine Kopier-Lizenz deshalb billig abgab.

5.2.3 Veröffentlichung

Ein Spezialfall des oben genannten Punktes d, der auch (aber nicht nur) für Hobby-Programmierer in Betracht kommt, ist die Veröffentlichung des Programms in einer Fachzeitschrift. Gegen ein bestimmtes Seitenhonorar (z.B. 200 DM pro Druckseite) überschreibt man einem (aus urheberrechtlichen Gründen wirklich nur einem!) Verlag das Recht der Verwertung von Programm und Begleittext. Bei seriösen Verlagen erhält man vor dem Abdruck ein Vertragsformular, das den Autor verpflichtet, die Arbeit nicht Dritten anzubieten, und den Verlag bindet, dem Autor vor dem Abdruck noch Korrekturen der redaktionell bearbeiteten Version zu ermöglichen und nach Erscheinen des Heftes das vereinbarte Seitenhonorar zu überweisen.

Es sei dringend angeraten, eine Arbeit nie gleichzeitig mehreren Verlagen anzubieten, da die weniger seriösen Text und Programm oft ohne Rückmeldung beim Autor abdrucken. Hat der Autor inzwischen jedoch bei einem anderen Verlag den Annahme-Vertrag unterschrieben, kann er wegen Vertragsbruchs mit einer Konventionalstrafe belangt werden.

Das Programm sollte in einer fotografisch reproduzierbaren Form, also als möglichst kontrastreiches Listing vorliegen (frisches Farbband) und eventuell gleichzeitig z.B. auf Diskette beigelegt werden, damit es in der Redaktion getestet werden kann (es gibt tatsächlich Zeitschriften, die Programme vor dem Abdruck testen!). Das Listing sollte ein möglichst kompaktes Format besitzen, d.h. die Zeilenlänge sollte entweder z.B. auf 40 Zeichen begrenzt oder von Haus aus etwa bei allen Zeilen gleich sein.

Liegen Skizzen bei (diese werden bei seriösen Verlagen ohnehin neu gezeichnet und müssen daher nur eindeutig lesbar sein), so sollten sie ebenso wie Listings im Begleittext mit ihrer numerischen Reihenfolge erwähnt und auf getrennten Blättern beigelegt sein, da sie einen anderen Bearbeitungsweg gehen als das Manuskript.

Der Text selbst beschreibt im Idealfall kurz, prägnant und sachlich Zweck, Funktionsweise und Bedienung des Programms. An seinem Ende sollten die Bildunterschriften (Listings, Skizzen, ev. Fotos) zusammengefaßt sein, da diese später erst einmontiert werden. Unnötige Ausschweifungen auf Rand- und Detailfragen sollte man ebenso vermeiden wie Fremdworte, für die auch deutsche Worte geläufig sind.

Ein wesentlicher Aspekt der Veröffentlichung ist, daß spätere Veröffentlichungen desselben Programms dem urheberrechtlichen Schutz und dem Verlagsrecht unterliegen. In der Praxis bedeutet das: Das Verwertungsrecht an Text, Skizzen und Listings geht ab dem Zeitpunkt des Erscheinens für ein Jahr an den Verlag über. Der Verlag kann z.B. das Programm innerhalb dieser Frist einem anderen Verlag (auch einem ausländischen) zum Nachdruck freigeben oder selbst in einer Sonderausgabe erneut veröffentlichen. Üblicherweise erhält der Autor jedoch in solchen Fällen nochmals eine Vergütung, die allerdings geringer ist als die für die ursprüngliche Veröffentlichung.

Der Verlag wird seinerseits ihm bekannt werdende Urheberrechtsverletzungen Dritter (z.B. Abkupfern des Programms in einer anderen Publikation ohne Wissen des Autors) verfolgen. Umgekehrt verpflichtet sich der Autor gegenüber dem Verlag, die Arbeit innerhalb eines Jahres nicht anderen Verlagen zum Abdruck anzubieten.

5.3 Kommerzielle Anwenderprogramme

5.3.1 Wordstar

Wordstar ist ein weit verbreitetes Textverarbeitungsprogramm von Micropro, das es für die Prozessorfamilien 8080/Z80 und 8088/8086 gibt. Zusätzlich gibt es vom gleichen Softwarehersteller auch Mailmerge zum Schreiben von Standard-Briefen an vorher eingegebene Adressensätze sowie Spellstar zur automatischen Kontrolle häufig vorkommender Worte auf korrekte Schreibweise im Text.

Wordstar wird von Diskette mit dem Filenamen WS geladen und gestartet. Die jeweils möglichen Kommandos werden im oberen Bildschirm-Drittel tabellarisch angezeigt; diese Übersicht kann aber auch ausgeblendet werden, so daß der gesamte Bildschirm zur Texteingabe frei ist. Die erstellten Textdateien können mit beliebiger Zeilenbreite entweder normal oder im Blocksatz (alle Zeilen werden durch Auffüllen mit Leerräumen gleich lang) ausgedruckt werden. Paßt bei der Eingabe des Textes ein Wort nicht mehr an das Ende der momentanen Zeile, so wird es ggf. dort „ausradiert" und in die nächste übernommen, so daß am Zeilenende keine Worte zerrissen werden.

Die folgende Übersicht gibt die Kommandos der Wordstar-Version 3.0 wieder. CTRL-Zeichen werden mit dem Zeichen ^ markiert, so ist z.B. für ^J die CTRL-Taste niederzuhalten und J anzutippen.

Hilfsmenüs

^J	Menü für Erläuterungen
^K	Block-Bearbeitungsbefehle
^O	Format-Anweisungen
^P	Druck-Kommandos
^Q	Text-Bearbeitungsbefehle

Steuerkommandos

D	Textdatei bearbeiten (Steuerzeichen mit Bit 7 = 1; vgl. N)
E	Dateinamen ändern (^KE)
F	Disk-Inhaltsverzeichnis ein/aus (^KF)
H	Hilfsstufe setzen (^JH)
L	Laufwerk ändern
M	Mailmerge starten
N	Programmdatei bearbeiten (vgl. D; vermeidet Bit 7 = 1)
O	Datei kopieren (^KO)
P	Datei ausdrucken (^KP)
R	Anderes Programm laden
S	Spellstar starten
X	Sprung zum Betriebssystem (z.B. CP/M)
Y	Datei löschen (^KJ)
^KD	Datei abspeichern, Bearbeitung beenden
^KQ	Bearbeitung der Datei abbrechen
^KS	Datei abspeichern
^KX	Datei abspeichern, Wordstar beenden
^QQ	Folgenden Befehl wiederholen
^U	Kommandoausführung abbrechen

Cursor-Steuerung

^A	Cursor ein Wort nach links
^C	Zur nächsten Seite springen
^D	Ein Zeichen nach rechts
^E	Eine Zeile nach oben
^F	Ein Wort nach rechts
^I	Tabulator-Sprung nach rechts (TAB)
^R	Zur vorherigen Seite springen
^S	Ein Zeichen nach links
^W	Bildschirm eine Zeile nach unten rollen
^X	Eine Zeile nach unten
^Z	Schirm eine Zeile nach oben rollen
^QB	Cursor zum Blockanfang
^QC	Cursor zum Dateiende

^QD	Zum rechten Rand
^QE	Zum oberen Bildschirmrand
^QF	Zeichenfolge suchen (weiter mit ^L)
^QK	Zum Blockende
^QP	Zur letzten Position
^QR	Zum Textbeginn
^QS	Zum linken Rand
^QV	Zum letzten Block
^QW	Bildschirm nach unten rollen
^QX	Zum unteren Bildschirmrand
^QZ	Bildschirm nach oben rollen
^Q0...^Q9	Zur Marke 0...9

Editier-Kommandos

^B	Zeilen neu umbrechen (formatieren)
^G	Zeichen unter dem Cursor löschen
^N	Zeile einfügen
^T	Wort rechts vom Cursor löschen
^V	Insert-Modus ein/aus
^Y	Zeile löschen
DEL	Zeichen links vom Cursor löschen
^QDEL	Bis zum Zeilenanfang löschen
^QA	Zeichenfolge ersetzen (weiter mit ^L)
^QY	Bis zum Zeilenende löschen

Ausgabe-Formatierung

^OC	Zeile zentrieren (Formatmitte)
^OD	Anzeige der Druckbefehle
^OE	Soft-Return ein/aus (d.h. Zeilenenden nicht mit CR kennzeichnen)
^OF	Die momentane Zeile wird als Formatzeile definiert
^OG	Absatz einrücken
^OH	Trennvorschlag am Zeilenende ein-/ausschalten
^OI	Tabulator setzen
^OJ	Blocksatz ein-/ausschalten
^OL	Setzen des linken Randes
^OP	Seiten-Anzeige ein-/ausschalten
^ON	Tabulator-Position löschen
^OR	Setzen des rechten Randes
^OS	Zeilenabstand einstellen
^OT	Formatzeile ein-/ausschalten
^OV	Umschaltung zwischen festen und variablen Tabulatoren
^OW	Wortumbruch (autom. Trennung am Zeilenende) ein/aus
^OX	Rand freigeben (um über ihn hinausschreiben zu können)

Kommandos für Textblöcke und Dateien

^KB	Markieren des Blockanfangs
^KC	Block kopieren
^KE	Umbenennen einer Datei
^KH	Blockmarkierung ein/aus
^KK	Markieren des Blockendes
^KN	Spaltenblock ein/aus
^KO	Kopieren einer Disk-Datei
^KR	Lesen eines Blocks (Textbausteins) von Diskette
^KV	Block verschieben
^KW	Block in Disk-Datei schreiben
^KY	Block löschen
^K0...^K9	Marken 0...9 ein/aus

Steuerung des Druckers

^PE,Q,R,W	Vom Anwender definierbare Steuerzeichen
^PA	Zeichengröße einstellen
^PB	Fettdruck Anfang/Ende
^PC	Pause im Ausdruck
^PD	Doppeldruck Anfang/Ende
^PF	Pseudo-Leerzeichen
^PG	Pseudo-Delete-Zeichen (Rubout)
^PH	Zeichen überdrucken
^PM	Zeile überdrucken (oder ^P Return)
^PN	Normale Zeichengröße
^PO	Leerzeichen
^PS	Unterstreichung Anfang/Ende
^PT	Hochzahl (Exponent)
^PV	Tiefzahl (Index) Anfang/Ende
^PX	Durchstreichen
^PY	Farbband-Umschaltung

Punktkommandos (keine CTRL-Zeichen nötig)

.BP	Bidirektionaler Druck
.CP	Bedingter Seitenumbruch
.CW	Schreibschrift-Raster (normal: .CW12 = 10 Zeichen/Zoll)
.FM	Platz für Fußnote schaffen (Seiten-Nr./#)
.FO	Fußnote
.HE	Überschrift
.HM	Rand für Überschrift
.IG	Kommentar (wird nicht gedruckt; auch „.." möglich)
.LH	Zeilenhöhe (je nach Drucker; normal: .LH8 = 6 Zeilen/Zoll)
.MB	Unterer Rand

.MT	Oberer Rand
.OP	Seitennummer nicht drucken
.PA	Neue Seite
.PC	Spalte für Seitennummer
.PL	Papierlänge einstellen
.PN	Seitennummer
.PO	Druckspalte
.SR	Zeichen hoch- oder tiefstellen
.UJ	Proportionalschrift (nur mit geeigneten Druckern möglich)

Mailmerge-Punktkommandos

.AV	Abfrage eines Variablenwerts
.CS	Schirm löschen/Meldung ausgeben
.DM	Meldung ausgeben
.DF	Datei als Daten-File deklarieren
.FI	Datei einfügen
.IJ	Blocksatz bei Eingabe
.LM	Linken Rand einstellen
.LS	Zeilenabstand einstellen
.OJ	Blocksatz bei Ausgabe
.PF	Druckzeilen-Formatierung
.RM	Rechter Rand
.RP	Wiederholung, bis alle Daten bearbeitet
.RV	Variablenwerte einlesen
.SV	Setzen einer Variablen

5.3.2 Applewriter-II

Applewriter-II (Version 1.1) ist ein 6502-Textverarbeitungs-Programm für den Apple-II und kompatible Computer. Die Texteingabe erfolgt fortlaufend (d.h. außer bei Absätzen ohne Druck auf Return am Zeilenende) mit Hilfe des 40-Zeichen-Bildschirms. Der Text wird ab Adresse hex 1901 abgelegt und als Binär-Datei auf Disk gespeichert. Dem verwendeten Namen wird dabei automatisch „TEXT." vorangestellt. Als Textende-Markierung dient das Byte hex 60. Bei der ursprünglichen Version war der im Speicher abgelegte Code noch nicht mit ASCII identisch, heute sind jedoch auch ASCII-konforme Versionen in Umlauf. Das Editor-Programm selbst beginnt bei der Adresse 0803 (ebenso das nachzuladende Druckprogramm).

Beim Start meldet sich das Editor-Programm zunächst mit einem Menü auf dem Bildschirm, das folgende Punkte umfaßt:

<E>	Editieren
<N>	Neuer Text
<L>	Von Disk laden
<S>	Auf Disk speichern
<D>	Text drucken
<B>	Programm beenden

Bei Eingabe von N wird nach einer weiteren Sicherheits-Abfrage (Text löschen: j/n) der Textspeicher neu initialisiert. Bei Druck auf D wird ein besonderes Druckprogramm von Disk geladen, das den fortlaufend eingegebenen Text in Zeilen zerlegt, indem es nach passenden Leerräumen sucht. Im Editor-Modus (Taste E bzw. N) stehen wesentlich weniger Kommandos als z.B. bei Wordstar zur Verfügung, was ein schnelles Erlernen begünstigt. Das Zeichen ^ markiert hier wieder, daß es sich um CTRL-Tastenkombinationen handelt:

^B	Cursor an den Textanfang setzen
^D	Disk-Kommando eingeben, z.B. CATALOG usw.
^E	Cursor ans Textende setzen
^F	Anzeige des noch freien Speicherplatzes
^I	Einfügen einer Disk-Datei an der Cursor-Position
^K	Kopieren des Textes ab Markierung auf Disk; Markierung = irgendeine Zeichenfolge, z.B. /ABC/
^P	Ein-/Ausschalten des Lautsprechers bei Zeilensprüngen
^S	Suchen und Ersetzen: /ABC/ = nur suchen, /ABC/DE/ = ABC durch DE ersetzen
^T	Halben Bildschirm nach oben
^V	Halben Bildschirm nach unten
^W	Wort links vom Cursor löschen (d.h. bis zum Leerraum)
^X	Absatz links vom Cursor löschen (d.h. bis Return)
^Y	Verschieben von Textteilen

Wenn man einmal auf die Escape-Taste drückt, wird der nachfolgend eingegebene Buchstabe in Großschrift abgespeichert. Diese „Krücke" ist nötig, weil die frühere Apple-II-Tastatur nicht zwischen Groß- und Kleinschrift umschalten konnte.

Ein zweimaliger Druck auf die Escape-Taste macht aus dem normalerweise weiß blinkenden Cursor ein inverses Pluszeichen. Er läßt sich dann mit den auf der Tastatur in Rautenform angeordneten Tasten J, K, I und M nach links, rechts, oben und unten bewegen. Mit CTRL-Q kann man auch zum Menü zurückkehren, um z.B. den erstellten Text auf Diskette zu speichern oder zu drucken. Ein Druck auf eine beliebige andere Taste schaltet auf den „Normal-Cursor" zurück. Mit dreimal Escape wird der Cursor als Prozent-Zeichen dargestellt, und man kann Sonderzeichen im Text eingeben.

Je nach Systemkonfiguration und Anpaßwünschen kann es nützlich sein, die Adressenbelegung des Applewriter-Programms zu kennen. Im folgenden werden daher einige wichtige Speicherzellen und Routinen des Editors genannt.

0000	Sprungvektor für CTRL-Kommandos (2 Byte)
0002	Flag für Cursor: „+“ wenn > 0
000B	Flag für Groß-/Kleinschreibung; Cursor „^“ wenn > 0
000C	Flag für Cursor; „%“ wenn > 0
0014	Textzeiger; 2 Byte (ab $1901)
0016	Pufferzeiger (abwärts ab $959E)
0803	Startadresse; Sprung nach $10F8
0806	UP: Zeichen von der Tastatur holen
0986	Tabelle mit CTRL-Zeichen (bis 0995)
09C1	Zeichen einlesen und decodieren
0A40	CTRL-Zeichen decodieren
0A60	Sprünge zu CTRL-Kommando-Routinen
0AD0	Cursor als „^“ darstellen
0AD6	Cursor als „+“ darstellen
0ADC	Cursor als „%“ darstellen
0FE8	Menütext (ASCII)
1825	UP: Zeichen im Video-RAM (0400...07FF) speichern
1855	Zeichen über Monitorprogramm ausgeben

Möglicherweise ist es interessant für Sie zu erfahren, daß die erste Auflage dieses Buches vollständig mit Hilfe einer an den ASCII-Standard angepaßten Applewriter-Version entstanden ist. Da eine „richtige“ Tastatur und, im Apple-kompatiblen Rechner, ein Zeichengenerator mit Groß- und Kleinschrift verwendet wurde, war die Kleinschrift-Krücke des Applewriter hierzu jedoch nicht nötig.

5.4 Algorithmen

Unter einem Algorithmus versteht man eine Vorschrift zur Lösung eines im engeren Sinne numerischen Problems, im weiteren Sinn auch eine Vorschrift zur Behandlung von Zeichenfolgen. In diesem Kapitel werden häufig benötigte Algorithmen vorgestellt, damit Sie sie bei Bedarf in eigene selbstgeschriebene Programme einbauen können.

5.4.1 Funktionen

Nur ein Teil der in der Mathematik üblichen Funktionen ist in den Interpretern und Compilern für höhere Programmiersprachen enthalten, z.B. Quadratwurzel (SQR in Basic, SQRT in Pascal), natürlicher Logarithmus (LOG, bei Taschenrech-

```
Hierarchie ()  ^  */  +-
           d.h. 30-4*2^3 entspricht 30-(4*(2^3)) = -2

Fakultät   X! = 1*2*3*...*X

Natürl.Log.LOG(X)=X-1-(X-1)^2/2+(X-1)^3/3-... für 0<X<=2
           LOG(X)=(X-1)/X+(X-1)^2/(2*X^2)+(X-1)^3/(3*X^3)+...
           für X>0.5

Tangens    TAN(X)=X+X^3/3+2*X^5/15+17*X^7/315+ ... für X<PI/2
           TAN(X)=SQR((1-COS(2*X))/(1+COS(2*X)))

Cotangens  COT(X)=1/TAN(X)

Sinus      SIN(X)=TAN(X)/SQR(1+(TAN(X)^2))
           SIN(X)=X-X^3/3!+X^5/5!-X^7/7!+ ...

Cosinus    COS(X)=1/SQR(1+(TAN(X)^2))
           COS(X)=1-X^2/2!+X^4/4!-X^6/6!+ ...

Secans     SEC(X)=SQR(1+(TAN(X)^2))

Cosecans   CSC(X)=SQR(1+(TAN(XX)^2))/TAN(X)

Arcussinus ASN(X)=ATN(X/SQR(1-X*X))

Arcus-Cos. ACS(X)=ATN(SQR(1-X*X)/X)

Hyperbel-S.SINH(X)=(E^X-E^-X)/2
Hyperbel-C.COSH(X)=(E^X+E^-X)/2
Hyperbel-T.TANH(X)=(E^X-E^-X)/(E^X+E^-X)
           mit E=2.71828 (Eulersche Konstante)

Hyp.-Cot.  COTH(X)=1/TANH(X)

Dekad.Log. LG(X)=0.4343...*LOG(X)  mit LOG=natürl. Log.
```

Abb. 5.4.1.1 Algorithmen zum Berechnen der wichtigsten Funktionen

nern errechnet die LOG-Taste dagegen meist den dekadischen Logarithmus!), Sinus (SIN), COS, TAN, ATN.

Abb. 5.4.1.1 zeigt, wie man seltener gebrauchte Funktionen mit den vorhandenen realisieren kann. Hierbei sollte man allerdings beachten, daß diese Ersatzlösungen oft nur bei einem eingeschränkten Wertebereich arbeiten. Enthalten sie z. B. eine Division, so darf der Divisor nicht Null werden. Ferner können sich Ungenauigkeiten der verwendeten Funktionen je nach Argument entweder zufällig aufheben oder aufsummieren.

Es sind sowohl Näherungsreihen zur Berechnung elementarer Funktionen als auch Möglichkeiten zum Ersetzen seltenerer angegeben. Innerhalb von Interpretern werden zur Berechnung von Funktionen allerdings meist nicht solche For-

meln, sondern Näherungspolynome und Koeffizienten-Tabellen verwendet, da diese bei ausreichender Genauigkeit eine höhere Geschwindigkeit erlauben.

5.4.2 Numerische Algorithmen

Natürlich können hier nicht alle denkbaren Algorithmen dargestellt werden. Einige typische seien aber als Basic-Routinen gezeigt, die auf einem Apple-II entstanden. Damit soll nicht gesagt werden, daß Basic die „optimale" Programmiersprache ist. Vielmehr wird dabei lediglich berücksichtigt, daß Basic nach wie vor am weitesten verbreitet ist. (Diese Vorbemerkung ist nötig, da oft eine unnötige Diskussion darüber angeheizt wird, welche Programmiersprache die „beste" ist. Hier geht es um Algorithmen, nicht um Sprachen.)

Primzahlen

Primzahlen sind Zahlen, die selbst nicht ohne Rest durch andere ganze Zahlen außer 1 teilbar sind, also 2, 3, 5, 7, 11, 13 usw. Ganze Zahlen, die nicht selbst Primzahlen sind, lassen sich stets in Primfaktoren zerlegen, z.B. ergibt sich die Zahl 30 aus der Multiplikation der Primfaktoren 2, 3 und 5.

Eine beliebte mathematische „Trockenübung" ist das Zerlegen einer Zahl in ihre Primfaktoren, normalerweise eine zeitraubende Tätigkeit. Wie so etwas in einem Programm erledigt werden kann, zeigt die *Abb. 5.4.2.1.* Das Unterprogramm ab Zeile 100 testet, ob die Zahl A durch eine Primzahl P teilbar ist, und gibt letztere gegebenenfalls als Primfaktor aus. Um Primzahlen p zu errechnen, die größer als 3 sind, wird folgender Algorithmus angewandt:

$p = 6n +/- 1$

Dabei ist n = 1, 2, 3 usw., also eine ganze Zahl. p nimmt also die Werte 5, 7, 11, 13, 17, 19, 23, 25 usw. an. Natürlich sind nicht alle so gewonnenen Zahlen wirklich

```
10HOME:PRINT"PRIMFAKTOR-ZERLEGUNG"
15PRINT"(ENDE: 0 ODER 1)"
20PRINT:INPUT"ZAHL: ";Z:I=1:PRINT
25IFZ<2THENPRINT"BASIC":END
30P=2:GOSUB100:P=3:GOSUB100:P=1
40P=P+4:GOSUB100
50P=P+2:GOSUB100:IFP*P<ZTHEN40
60IFZ>1THENP=Z:GOSUB110
65IFI<3THENPRINT"EINE PRIMZAHL!"
70GOTO20
100A=Z/P:IFA<>INT(A)THENRETURN
110Z=A:PRINT"FAKTOR ";I;":";
120HTAB12:PRINTP:I=I+1:GOTO100
```

Abb. 5.4.2.1
Basic-Programm zum Zerlegen einer Zahl in ihre Primfaktoren

```
10DIMP%(999):P%(1)=2:P%(2)=3:N=2:HOME
15PRINT"PRIMZAHLEN-BERECHNUNG":PRINT
20INPUT"OBERES LIMIT? ";L:PRINT:Z=1
22IFL<5THENPRINT"BASIC":END
25PRINT"ZAHL 1: 2",:PRINT"ZAHL 2: 3",
30Z=Z+4:GOSUB50:Z=Z+2:GOSUB50
35IFZ<L-3THEN30
40END
50FORI=1TON:IFP%(I)*P%(I)>ZTHEN80
60A=Z/P%(I):IFA=INT(A)THENRETURN
70NEXT
80N=N+1:P%(N)=Z:PRINT"ZAHL ";N;": ";Z,
90RETURN
```

Abb. 5.4.2.2
Berechnen von Primzahlen. Da sich das Programm die bisher errechneten Primzahlen merkt, ist eine hohe Geschwindigkeit erzielbar

Primzahlen (z.B. 25). Die Methode ist jedoch effizienter, als alle natürlichen Zahlen (also 1, 2, 3 usw.) darauf testen zu müssen, ob sie Primzahlen sind oder nicht.

Man kann Primzahlen auch automatisch fortlaufend errechnen lassen. Das Programm in *Abb. 5.4.2.2* wendet dabei, um Zeit zu sparen, einen Trick an: Es merkt sich nämlich alle bisher errechneten Primzahlen im Feld P%. Dadurch ist es schneller möglich, eine Zahl in ihre Primfaktoren zu zerlegen und somit zu testen, ob sie selbst eine Primzahl ist. Auch hier wird der gleiche Algorithmus wie im obigen Beispiel angewandt, lediglich die Primzahlen 2 und 3 werden davon nicht erfaßt. Es ist natürlich auch nicht nötig, jede zu prüfende Zahl durch sämtliche niedrigeren Zahlen zu teilen; es genügt, wenn die Werte bis zur Quadratwurzel der neuen Zahl als Divisoren getestet werden.

Rollen-Berechnungen

Wie lang ist ein Tonband einer bekannten Dicke auf einer Rolle mit bekanntem Innen- und Außendurchmesser? Oder wie groß wird der Spulendurchmesser bei einer bestimmten Länge des Tonbandes? Diese und ähnliche Rechenaufgaben löst das Applesoft-Basic-Programm in *Abb. 5.4.2.3.*

Als „Hülsendurchmesser" wird hier der Innendurchmesser der Rolle bezeichnet. Die eigentliche Berechnung findet jeweils in einer einzigen Formel-Zeile statt (140 bzw. 240).

Tage zwischen Daten und Biorhythmus

Ein häufig vorkommendes Problem ist es, die Anzahl der Tage zwischen zwei Daten ausrechnen zu müssen. Das Programm in *Abb. 5.4.2.4* umkleidet den dafür nötigen Algorithmus gleich mit etwas Unterhaltsamem, nämlich mit der Berechnung des sogenannten Biorhythmus. Darunter versteht man regelmäßig wiederkehrende Zyklen, denen nach Ansicht einiger Wissenschaftler das menschliche Leben unterliegt. Die Zyklendauer ist je nach Kriterium unterschiedlich:

```
10PI=3.1415926:HOME:PRINT"BESTIMMUNG VON BAHNLAENGEN":PRINT
20PRINT"GESUCHTE GROESSE:"
30PRINT:PRINT"1 = BAHNLAENGE":PRINT"2 = ROLLENDURCHMESSER"
40PRINT"Q = PROGRAMM BEENDEN":PRINT
50GETA$:PRINTA$:PRINT:IFA$="Q"THENEND
60IFA$="1"THEN100
70IFA$="2"THEN200
90RUN
100REM BAHNLAENGE
110INPUT"MATERIALDICKE (MUE): ";MD
120INPUT"HUELSENDURCHM. (MM): ";HD
130INPUT"ROLLENDURCHM.  (MM): ";RD
140BL=INT(PI/(4*MD)*((RD*RD)-(HD*HD)))
150PRINT:PRINT"BAHNLAENGE = ";BL;" METER"
160GOTO260
200REM ROLLENDURCHMESSER
210INPUT"HUELSENDURCHMESSER (MM): ";HD
220INPUT"BAHNLAENGE      (METER): ";BL
230INPUT"MATERIALDICKE     (MUE): ";MD
240RD=INT(SQR((BL*MD*4/PI)+HD*HD))
250PRINT:PRINT"ROLLENDURCHMESSER = ";RD;" MM"
260PRINT:PRINT"BITTE DRUECKEN SIE EINE TASTE!";:GETA$:RUN
```

Abb. 5.4.2.3
Errechnen von Bahnlängen und Rollendurchmessern z. B. von Tonband-Spulen

```
5DIMM(12):HOME:PRINT"BIORHYTHMUS":PRINT
10INPUT"GEBURTSTAG (T,M,J):    ";T,M,J
20GOSUB100:A=S:RESTORE
30INPUT"PROGNOSE FUER (T,M,J): ";T,M,J
40GOSUB100:B=S:PRINT:RESTORE
50PRINT"VERFASSUNG AM "T"."M"."J":"
60L=28:PI=3.14159:GOSUB90:PRINT
70PRINT"SEELISCH: "INT(N)"%"
80L=33:GOSUB90
82PRINT"GEISTIG:  "INT(N)"%"
84L=23:GOSUB90
86PRINT"PHYSISCH: "INT(N)"%"
88PRINT:PRINT"NOCH'N DATUM (J/N)? ";:GETA$:PRINTA$:IFA$<>"J"THENEND
89GOTO30
90N=100*SIN(2*PI*(B-A)/L):RETURN
100Z=0:IFJ/4<>INT(J/4)THEN300
140IFM<=2THEN300
290Z=1
300Z=Z+(J-1)*365+INT((J-1)/4)
310FORK=1TO12:READM(K):NEXT
340S=Z+M(M)+T:RETURN
500DATA0,31,59,90,120,151,181,212,243,273,304,334

>RUN
BIORHYTHMUS

GEBURTSTAG (T,M,J):    16,3,1952
PROGNOSE FUER (T,M,J): 4,5,1985

VERFASSUNG AM 4.5.1985:

SEELISCH: 97%
GEISTIG:  -99%
PHYSISCH: 88%

NOCH'N DATUM (J/N)? N
```

Abb. 5.4.2.4
Dieses Programm ermittelt die Zahl der Tage zwischen zwei Daten und daraus den Biorhythmus

Seelisch: 28 Tage
Geistig: 33 Tage
Physisch: 23 Tage

Alle drei Zyklen beginnen am Tag der Geburt im Nullpunkt und werden üblicherweise im Bereich −100...+100 % angegeben.

Zum abgedruckten Programm ist zu bemerken, daß es nur innerhalb eines „normalen" Lebensalters korrekt arbeitet, was mit der komplizierten Zuweisung von Schaltjahren zusammenhängt:

1. Jedes durch 4 teilbare Jahr ist ein Schaltjahr.
2. Jedes durch 100 teilbare Jahr ist jedoch keins.
3. Dagegen ist ein durch 400 teilbares Jahr doch eins.

Im vorliegenden Programm wird nur Punkt 1 berücksichtigt. In Zeile 500 enthält es jene Tage im Jahr, in denen ein neuer Monat beginnt. Nach RUN ist ein Probelauf wiedergegeben.

ISBN-Prüfprogramm

Die „Internationale Standard-Buch-Nummer" (ISBN) dient zur internationalen Kennzeichnung von Büchern, damit jeder Buchhändler schnell anhand von Tabellen herausfinden kann, in welchem Land und von welchem Verlag das Buch herausgegeben wurde.

Damit bei der Übermittlung oder beim Abschreiben der ISBN Fehler sofort entdeckt werden können, ist die letzte Ziffer der insgesamt zehnstelligen Zeichenfolge einem Prüfzeichen vorbehalten. Es entsteht durch bewertetes Aufsummieren der vorangehenden Ziffern modulo 11; ist das Ergebnis 10, so wird (um mit einem Zeichen auszukommen) ein X geschrieben.

Die Routine in *Abb. 5.4.2.5* erlaubt die Eingabe einer ISBN, prüft zunächst die korrekte Länge von zehn Zeichen und vergleicht die eingegebene Prüfziffer mit der zum Vergleich selbst errechneten.

```
10REM --- ISBN-PRUEFPROGRAMM ---
20PRINT:INPUT"BUCHNUMMER: ";IS$
30IFLEN(IS$)<>10THENEND
40FORI=1TO9:Z(I)=VAL(MID$(IS$,I,1))
50NEXT:E=0:FORI=1TO9:P=Z(I)*(11-I)
60E=E+P:NEXT
70IFE/11=INT(E/11)THENZ$="0":GOTO100
80Z=11*(INT(E/11)+1)-E:Z$=STR$(Z)
90IFZ=10THENZ$="X"
100IFZ$=RIGHT$(IS$,1)THEN120
110PRINT"FALSCHE EINGABE!":GOTO20
120PRINT"PRUEFZIFFER OK!":GOTO20
```

Abb. 5.4.2.5
Prüfprogramm für „Internationale Standard-Buch-Nummern" (ISBN)

```
10INPUT"GESPARTER BETRAG PRO JAHR:";R
20INPUT"ANFANGS-KAPITAL:          ";K
30INPUT"ZINSSATZ IN PROZENT:      ";P
40INPUT"ZEITRAUM IN JAHREN:       ";N
50Q=1+P/100:A=(R*Q*(Q^N-1))/(Q-1)
60PRINT:PRINT"ENDKAPITAL: ";:K=K*Q^N+A
70PRINTINT(K*100)/100:END

>RUN
GESPARTER BETRAG PRO JAHR:1200
ANFANGS-KAPITAL:          500
ZINSSATZ IN PROZENT:      6.5
ZEITRAUM IN JAHREN:       5

ENDKAPITAL: 7961.51
```

Abb. 5.4.2.6
Zinseszins-Berechnung für Sparkonten — oder Schulden!

Zinseszins-Berechnung

Ausgehend von einem bestimmten Anfangskapital und einer jährlichen, konstanten Einzahlung kann man mit Hilfe des Programms in *Abb. 5.4.2.6* errechnen, wieviel sich auf einem Sparkonto angesammelt hat. Ebenso sind natürlich auch negative Beträge bei der Errechnung von Schuldzinsen möglich.

Das Endkapital wird in Zeile 70 mit einem einfachen Algorithmus auf ganze Pfennige, also auf zwei Stellen nach dem Komma, gerundet, da eine genauere Angabe nur verwirrend und unnütz wäre.

Statistische Berechnungen

Die wichtigsten statistischen Funktionen sind in *Abb. 5.2.4.7* enthalten. Die Varianz (Streuung) gibt dabei an, wie sehr die einzelnen eingegebenen Werte im Schnitt von ihrem Mittelwert abweichen. Die Standardabweichung ist die Wurzel aus der Varianz, der Standardfehler berücksichtigt auch die Zahl der Werte. Der Variations-Koeffizient ist der Quotient aus Standardabweichung und Mittelwert.

Abb. 5.4.2.8 zeigt schließlich ein Programm zur Berechnung des Korrelations-Koeffizienten. Er gibt an, wie gut sich mehrere Wertepaare x und y an eine Gerade anpassen lassen. Wenn alle Wertepaare auf einer Geraden liegen, wird der Korrelations-Koeffizient 1.

Zufallszahlen

Häufig benötigt man Zufallszahlen in einem vorgegebenen Wertebereich. In *Abb. 5.4.2.9* ist gezeigt, wie das bewerkstelligt werden kann, hier für ganzzahlige Ergebniswerte.

In manchen Fällen (z.B. Lottozahlen) ist es wünschenswert, daß sich die ausgegebenen Zufallszahlen nicht wiederholen. In diesem Fall muß man die Routine so erweitern, daß sie sich in einem Feld (Array) „merkt", welche Zahlen schon einmal vorkamen, und ggf. eine neue Zahl errechnet.

```
100PRINT"STATISTIK":PRINT
110PRINT"EINGABEENDE MIT 9999!":PRINT
120S=0:S1=0:N=0:REM SUMMEN UND ANZAHL
130INPUT"WERT: ";X:IFX=9999THEN150
140S=S+X:S1=S1+X*X:N=N+1:GOTO130
150M=S/N:V=(N*S1-S*S)/N/(N-1)
160S3=SQR(V):S4=SQR(V/N):Q=S3/M
170PRINT"WERTEZAHL:        ";N
180PRINT"SUMME DER WERTE: ";S
190PRINT"QUADRATSUMME:     ";S1
200PRINT"ARITHM.MITTELW.: ";M
210PRINT"VARIANZ:          ";V
220PRINT"STANDARD-ABW.:    ";S3
230PRINT"STANDARDFEHLER:   ";S4
240PRINT"VAR.-KOEFFIZIENT:";Q:END

>RUN
STATISTIK

EINGABEENDE MIT 9999!

WERT: 345.2
WERT: 9867
WERT: 567
WERT: 5428
WERT: 9999
WERTEZAHL:         4
SUMME DER WERTE:  16207.2
QUADRATSUMME:     127261525
ARITHM.MITTELW.:  4051.8
VARIANZ:          20531064
STANDARD-ABW.:    4531.12172
STANDARDFEHLER:   2265.56086
VAR.-KOEFFIZIENT:1.11829847
```

Abb. 5.4.2.7
Errechnen der wichtigsten Statistik-Werte einer Zahlenkolonne

```
100PRINT"KORRELATIONS-KOEFFIZIENT":PRINT
110S1=0:S2=0:S3=0:S4=0:S5=0
120INPUT"ZAHL DER WERTEPAARE: ";N
130FORI=1TON:PRINT"X";I;:INPUTX
140PRINT"Y";I;:INPUTY:S1=S1+X
150S2=S2+Y:S3=S3+X*Y:S4=S4+X*X
160S5=S5+Y*Y:NEXT:A=N*S3-S1*S2
170B=SQR((N*S4-S1*S1)*(N*S5-S2*S2))
180PRINT"KORRELATIONSKOEFFIZIENT=";A/B

>RUN
KORRELATIONS-KOEFFIZIENT

ZAHL DER WERTEPAARE: 4
X1?-4
Y1?6
X2?5.8
Y2?6.3
X3?6.1
Y3?-5
X4?6
Y4?7
KORRELATIONSKOEFFIZIENT=-.299258354
```

Abb. 5.4.2.8
Programm zur Berechnung des Korrelations-Koeffizienten einer Gruppe von Wertepaaren

```
100PRINT"ZUFALLSZAHLEN":PRINT
110INPUT"BEREICH: VON,BIS? ";MIN,MAX
120INPUT"ANZAHL DER ZAHLEN:";N
130FORI=1TON
140Z=INT((1+MAX-MIN)*RND(1)+MIN)
150PRINTZ:NEXT:END

>RUN
ZUFALLSZAHLEN

BEREICH: VON,BIS? 5,9
ANZAHL DER ZAHLEN:4
6
8
7
7
```

Abb. 5.4.2.9 Dieses Programm liefert Zufallszahlen in einem vorgebbaren Wertebereich

```
10GOTO500:REM FUER TEST
20:
100REM ADDITION
110X=X1+X2:Y=Y1+Y2:RETURN
190:
200REM SUBTRAKTION
210X=X1-X2:Y=Y1-Y2:RETURN
290:
300REM MULTIPLIKATION
310X=X1*X2-Y1*Y2
320Y=X1*Y2+Y1*X2:RETURN
390:
400REM INVERSES 1/(X1,Y1)
410T=X1*X1+Y1*Y1:X=X1/T
420Y=-Y1/T:RETURN
490:
500REM DEMONSTRATIONSPROGRAMM
510INPUT"X1,Y1,X2,Y2: ";X1,Y1,X2,Y2
530PRINT:PRINT"RESULTATE:   RE, IM"
540GOSUB100:PRINT"SUMME:        "X", "Y
550GOSUB200:PRINT"DIFFERENZ:    "X", "Y
560GOSUB300:PRINT"PRODUKT:      "X", "Y
570GOSUB400:PRINT"1/(X1,Y1):    "X", "Y
580END

>RUN
X1,Y1,X2,Y2: 2, 0, 5, 3

RESULTATE:    RE, IM
SUMME:        7, 3
DIFFERENZ:    -3, -3
PRODUKT:      10, 6
1/(X1,Y1):    .5, 0
```

Abb. 5.4.2.10 Komplexe Arithmetik mit Hilfe einiger Basic-Unterprogramme

Komplexe Arithmetik

Das Rechnen mit komplexen Zahlen der Form a+bj ist insbesondere in der Elektrotechnik oft erforderlich, wenn es um das Verhalten induktiver oder kapazitiver Bauelemente geht. Addition, Subtraktion, Multiplikation und Kehrwert (Inverses, daraus läßt sich bei Bedarf auch die Division ableiten) lassen sich mit den Routinen in *Abb. 5.4.2.10* realisieren.

Die einzelnen Routinen verlangen den Realteil der Eingabewerte in X1 und X2, den Imaginärteil in Y1 und Y2 (beim Kehrwert werden nur X1 und Y1 benötigt). Der reale Ergebnisteil steht dann in X und der imaginäre in Y.

5.4.3 String-Routinen

Nicht alle Computer besitzen so komfortable String-Befehle wie INSTR in MBasic. Ab Zeile 100 in *Abb. 5.4.3.1* steht deshalb ein Unterprogramm, das diesen Befehl simuliert. Es ermittelt die Position P von S2$ in S1$. Ist S2$ nicht in S1$ enthalten, wird P gleich Null gesetzt.

```
50REM *** STRING-ROUTINEN (APPLESOFT-BASIC) ***
55REM VARIABLEN: S1$,S2$ = ARBEITSSTRINGS
60REM P=POSITION IM STRING, T$=TEMPORAER-STRING
99:
100REM *** P=INSTR(S1$,S2$) ***
102FORP=1TOLEN(S1$)-LEN(S2$)+1
105IFMID$(S1$,P,LEN(S2$))=S2$THENRETURN
110NEXT:P=0:RETURN:REM 0 WENN S2$ NICHT IN S1$
120:
130REM *** MID$(S1$,P)=S2$ ***
132S1$=LEFT$(S1$,P-1)+S2$+MID$(S1$,P+LEN(S2$))
134RETURN:REM S2$ ERSETZT S1$ AB POS. P
140:
150REM *** S2$ IN S1$ LOESCHEN ***
152GOSUB100:IFP=0THENRETURN
153T$=MID$(S1$,P+LEN(S2$)):IFP<2THENS1$=T$:GOTO152
154S1$=LEFT$(S1$,P-1)+T$:GOTO152
155:
160REM *** S2$ NACH P IN S1$ EINFUEGEN ***
162IFP<1THENS1$=S2$+S1$:RETURN
164S1$=LEFT$(S1$,P)+S2$+MID$(S1$,P+1):RETURN
168:
170REM *** P IN S1$ KOMMABUENDIG AUSGEBEN ***
172S1$=STR$(P):S2$=".":GOSUB100
174S2$=S1$:IFP=0THENP=LEN(S2$)+1
175S1$="            ":P=LEN(S1$)-P:GOTO130
177REM NICHT FUER ZAHLEN MIT EXPONENT!
```

Abb. 5.4.3.1 Routinen zur Stringbehandlung in Applesoft-Basic

In MBasic ist es im Gegensatz zu vielen anderen Basic-Dialekten möglich, MID$ auf der linken (!) Seite einer Gleichung zu schreiben, um einen Teil eines Strings durch einen anderen zu ersetzen. Auch dieser Befehl kann notfalls in Basic simuliert werden, wie ab Zeile 130 gezeigt wird. Ab der Position P wird S1$ mit S2$ überschrieben, wobei für die Zahl der geänderten Zeichen in S1$ die Länge von S2$ ausschlaggebend ist; u. U. wird S1$ dabei also auch länger.

Manchmal möchte man bestimmte Zeichen oder Zeichenfolgen aus einem String entfernen. Die Routine ab Zeile 150 löscht den Teil von S1$, der gleich der Zeichenfolge in S2$ ist. Dabei kann S2$ auch wiederholt in S1$ auftreten. Das Unterprogramm verwendet die oben erwähnte Instring-Routine, um die Position von S2$ in S1$ zu ermitteln, T$ dient als String-Zwischenspeicher.

Umgekehrt ist es auch leicht möglich, in den String S1$ ab einer Position P die Zeichenfolge S2$ einzufügen, wobei natürlich S1$ entsprechend länger wird. Das bewerkstelligt die Routine ab Zeile 160.

Mit dem PRINT-USING-Befehl von MBasic ist es möglich, Zahlen in mehreren Zeilen so auszugeben, daß ihre Dezimalpunkte jeweils genau untereinander stehen. Ab Zeile 170 wird gezeigt, wie auch das mit einigen Anweisungen simuliert werden kann. Hierzu wird zunächst mit dem Instring-Unterprogramm die Position des Dezimalpunkts im Zahlenstring S1$ ermittelt, der mit STR$ aus P (der auszugebenden Zahl) ermittelt wird. Enthält S1$ keinen Punkt, war P also eine ganze Zahl, so nimmt ihn das Programm hinter der letzten Ziffer an. Nun wird eine Kette von Leerräumen als Platzhalter für den ganzzahligen Anteil von P mit dem Zahlenstring überschrieben. Dazu erfolgt ein Unterprogramm-Sprung nach Zeile 130.

Ein Programm, das ebenfalls die Position eines Strings in einem anderen ermittelt, zeigt *Abb. 5.4.3.2.* Der zu suchende String ist hier allerdings nur ein Zeichen lang: Das Programm dient der Decodierung von Kommandos z. B. nach Anzeige eines Menüs, damit der Benutzer ein bestimmtes Unterprogramm auswählen kann. Die zulässigen Kommando-Tasten stehen im String K$ in Zeile 30. Zeile 60 sucht nach der Position I des eingetippten Zeichens in K$. Ist es in K$ enthalten, so wird es auf den Bildschirm ausgegeben und das zugehörige Unterprogramm mit ON I GOSUB angesprungen.

Die beschriebene Methode der Kommando-Decodierung beinhaltet übrigens zwei heikle Punkte. Zum einen erfolgt, wenn T$ in K$ gefunden wurde, mit GOTO

```
20REM *** KOMMANDO-DECODIERUNG ***
30PRINT"KOMMANDO:";:K$="QABC"
50GETT$:FORI=1TOLEN(K$)
60IFT$<>MID$(K$,I,1)THENNEXT:GOTO50
70PRINTT$:ONIGOSUB80,100,200,300:GOTO30
80END
100PRINT"ROUTINE A":RETURN
200PRINT"ROUTINE B":RETURN
300PRINT"ROUTINE C":RETURN
```

Abb. 5.4.3.2
Nach der Anzeige eines Menüs kann diese Routine je nach gedrückter Taste bestimmte Unterprogramme aufrufen

ein Aussprung aus einer FOR-NEXT-Schleife, wobei der Return-Stack des Interpreters nicht zurückgesetzt wird. Das ist bei den meisten Interpretern nur zulässig, wenn vorher nicht schon eine andere FOR-Schleife nicht geschlossen wurde. Wenn später eine FOR-Schleife mit der gleichen Index-Variablen I geöffnet wird, so wird jedoch der Stack automatisch bereinigt, also zurückgesetzt. Ebenso ist normalerweise das Beenden des Programms (hier mit dem Kommando Q) durch Ansprung einer END-Zeile mit GOSUB unproblematisch, da ein erneutes RUN ebenfalls den Stack bereinigt.

5.4.4 Sortierverfahren

Die folgenden Verfahren zum Sortieren lassen sich gleichermaßen für numerische Daten und Zeichenfolgen (Strings) anwenden. Ebenso kann man durch Austausch meist nur eines Vergleichs-Operators zwischen auf- und absteigender Ordnung wählen.

Lineares Einfügen

Das lineare Einfügen ist eines der wohl naheliegendsten Sortierverfahren. Anhand einer alphabetisch geordneten Adressenkartei kann man es sich leicht vorstellen: Man sieht einfach nach, zwischen welche zwei Karteikarten die neue hineingesteckt werden muß.

Der Algorithmus hat also folgende Form: Man nehme ein Datenelement (beginnend beim ersten) und durchsuche die vorhandene Liste ab der momentanen Position, zwischen welche zwei anderen Elemente das aktuelle hineingehört. Das Element wird dann an der ursprünglichen Stelle gelöscht und an der neuen Position eingefügt.

Das Verfahren ist alles andere als schnell und erfordert auch ein umständliches Durchsuchen der Liste nach der richtigen Einfügungs-Stelle.

Binäres Einfügen

Statt die gesamte Folge von Elementen zu durchsuchen, kann man die richtige Einfügungsstelle auch finden, indem man zuerst in der Mitte der Datei nachsieht, ob die Stelle davor oder dahinter liegt. Liegt sie in der vorderen Hälfte, so kontrolliert man, ob die richtige Stelle im ersten oder zweiten Viertel der Datei liegt, und so weiter. Insbesondere, wenn die Daten teilweise schon sortiert sind (z. B. beim nachträglichen Einfügen neuer Daten in eine Liste), ist dieses Verfahren deutlich schneller als das lineare Einfügen.

Das binäre Einfügen ist auch das Verfahren, das der Mensch selbst typischerweise beim Einfügen neuer Karten in eine alphabetische Kartei verwendet.

```
10I=0:REM WERTE FUER TEST EINLESEN
20INPUTA$(I):IFA$(I)>="0"THENI=I+1:GOTO20
30I=I-1:GOSUB100:REM SORTIEREN
40FORJ=0TOI:PRINTA$(J):NEXT:END
50:
90REM *** BUBBLESORT IM FELD A$(J) ***
100F=0:FORJ=0TOI-1:REM I=LETZTES ELEMENT IN A$(J)
110IFA$(J+1)<A$(J)THENM$=A$(J+1):A$(J+1)=A$(J):A$(J)=M$:F=1
120NEXT:IFF=1THEN100
130RETURN
```

Abb. 5.4.4.1 Bubblesort-Programm in Applesoft-Basic zum alphabetischen Sortieren des Feldes A$ bis zum Index I. Für geringe Datenmengen ist das Verfahren meist ausreichend schnell

Bubblesort

Bubblesort heißt „Blasen-Sortieren". Man kann sich den Sortiervorgang so vorstellen wie das Aufsteigen von Blasen in einem mit Mineralwasser gefüllten Glas: Die Blasen verdrängen das Wasser, bis sie oben sind.

Das Verfahren beruht auf dem zyklischen Vertauschen der Datenelemente. *Abb. 5.4.4.1* zeigt ein in Applesoft-Basic geschriebenes Bubblesort-Programm, das das String-Feld A$ ab dem Index 0 bis zum Index I alphabetisch sortiert. Damit Sie das Programm gleich ausprobieren können, enthalten die Zeilen 10 bis 40 eine Eingabe- und Ausgabe-Routine. Es können mehrere Worte (jeweils mit Return abgeschlossen) eingegeben werden. Die Eingabe wird beendet, wenn man statt dessen z.B. nur einen Punkt eintippt. Dann erhält man eine sortierte Liste der Worte.

Das Programm merkt sich in der Variablen F (Flag), ob in einem Durchlauf überhaupt noch eine Vertauschung nötig war; wenn nicht, ist das Feld fertig sortiert, und die Routine wird beendet. (In Zeile 110 könnte man zur Vertauschung von zwei Feldelementen auch den MBasic-Befehl SWAP einsetzen, falls dieser zur Verfügung steht.)

Bubblesort ist zwar sehr einfach zu programmieren, aber relativ langsam. Insbesondere wenn am Ende der Datei ein Element steht, das an den Anfang gehört, sind sehr viele Vertauschungszyklen nötig, ebenso im umgekehrten Fall, wenn ein am Anfang stehendes Element an den Schluß sortiert werden muß.

Shakersort

Aus Bubblesort läßt sich ein Verfahren entwickeln, das das asymmetrische Verhalten beseitigt. Shakersort fängt nicht bei jedem Durchlauf von vorn an, die Elemente zu vertauschen, sondern abwechselnd von vorn und hinten. Dadurch wird insbesondere bei solchen Dateien, die schon teilweise sortiert sind, ein Zeitgewinn gegenüber Bubblesort erzielt, der allerdings nicht sehr groß ist. Insbesondere kann Shakersort gegenüber den nachfolgend besprochenen Methoden nicht mithalten, da das Sortieren nach wie vor durch einfaches Vertauschen aufeinanderfolgender Elemente erfolgt.

Heapsort

Alle bisher besprochenen Sortierverfahren bearbeiten bei jedem Durchlauf jeweils nur ein einziges Element der Liste. Das setzt ihrer Geschwindigkeit eine natürliche Grenze. Verwendet man dagegen ein numerisches Hilfsfeld (Heap), in dem man sich bei einem Durchlauf die Sortierordnung gleich mehrerer Elemente merken kann, ergibt sich eine erhebliche Geschwindigkeits-Erhöhung. Der Heap läßt sich als binärer Baum auffassen (siehe Kapitel über Datenstrukturen), in dem die Elemente entsprechend ihrer Ordnung stehen.

Ein Sortier-Durchlauf geht bei Heapsort in zwei Schritten vor sich: Zunächst wird durch Vergleichen der Listen-Elemente der Heap als Binärbaum aufgebaut. Dann kann man durch Abarbeiten dieses Baums Element für Element an die richtige Stelle der Liste einbauen. (Das Verfahren ähnelt also ein wenig dem „binären Einfügen", allerdings wird hier nicht die Datei selbst, sondern der Heap als Hilfsliste zum Ermitteln der richtigen Einfügungsstelle herangezogen.)

Heapsort hat allerdings an Bedeutung verloren, nachdem C. A. Hoare ein noch schnelleres Verfahren namens Quicksort entwickelte.

Quicksort

Insbesondere bei langen Dateien ist Quicksort heute ein Standard-Verfahren. Es arbeitet ebenfalls mit einem numerischen Hilfsfeld, beginnt jedoch mit dem Sortieren normalerweise nicht einfach mit dem ersten Element der Datei, sondern z.B. mit dem mittleren. Die Datei wird nun sowohl vom Anfang als auch vom Ende her durchsucht, bis vor dem einzuordnenden Element ein größeres und hinter ihm ein kleineres gefunden ist. Diese beiden Elemente werden nun vertauscht. Der Vorgang wiederholt sich, bis sich die Indizes irgendwo in der Mitte treffen. Das Resultat sind dann zwei Dateihälften, wobei die linke kleinere und die rechte größere Elemente als das in der Mitte befindliche enthält. Beide Hälften werden nun ge-

```
272REM *** QUICKSORT IM FELD A$(I) ***
276I=I-1:L=1:N(L)=I+1:M=0
278J=N(L):K=M-1:IFJ-M<3GOTO300
280M1=INT((K+J)/2)
282K=K+1:IFK=JGOTO292
284IFA$(K)=<A$(M1)GOTO282
286J=J-1:IFK=JGOTO292
288IFA$(J)>=A$(M1)GOTO286
290Y$=A$(K):A$(K)=A$(J):A$(J)=Y$:GOTO282
292IFK>=M1THENK=K-1
294IFJ=M1GOTO298
296Y$=A$(K):A$(K)=A$(M1):A$(M1)=Y$
298L=L+1:N(L)=K:GOTO278
300IFJ-M<2GOTO306
302IFA$(M)<A$(M+1)GOTO306
304Y$=A$(M):A$(M)=A$(M+1):A$(M+1)=Y$
306M=N(L)+1:L=L-1:IFL>0GOTO278
308PRINT"**** SORTIERT ****":RETURN
```

Abb. 5.4.4.2
Quicksort-Routine zum alphabetischen Sortieren des Feldes A$ bis zum Index I-1

trennt nach dem gleichen Verfahren sortiert. Die Datei wird schließlich solange weiter zerlegt, bis jeder Teil nur noch aus einem Element besteht. Da schon das Sortieren von zwei Dateien von z.B. je 50 Zeilen viel schneller geht als das Sortieren einer einzigen mit 100 Zeilen Umfang, ist Quicksort in der Regel wesentlich schneller als andere Verfahren.

Ein Basic-Programm, das das String-Feld A$ nach dem Quicksort-Verfahren alphabetisch sortiert, zeigt *Abb. 5.4.4.2.* Vor dem Aufruf müssen A$ und das Hilfsfeld N mit DIM ausreichend dimensioniert werden. Als Anhaltspunkt kann man annehmen, daß mit n Elementen des Hilfs-Arrays N etwa 2↑(n/2) Elemente von A$ sortiert werden können, so daß z.B. DIM A$(500), N(20) sinnvoll wäre. Im Beispielprogramm werden I Elemente von A$ (inklusive Null als Index) sortiert, so daß I auf das dem letzten besetzten folgende Element zeigt.

5.4.5 Grafik-Algorithmen

Viele Computer besitzen nur sehr einfache Befehle zur grafischen Ausgabe, in vielen Fällen ist lediglich das Setzen einzelner Punkte unter Angabe ihrer Koordinaten möglich. Linien, Kreise und andere Formen müssen dann daraus abgeleitet werden. Ebenso ist natürlich die Situation beim Erstellen eines in Assembler geschriebenen Betriebssystems für einen grafikfähigen Rechner. Die wichtigsten Grundformen grafischer Elemente werden daher im folgenden kurz beschrieben.

Gerade Linien

Am häufigsten benötigt man wohl Linien zwischen zwei Punkten. Im kartesischen Koordinatensystem *(Abb. 5.4.5.1)* kann man die Lage der Punkte durch x- und y-Werte definieren, bei P1 durch X1 und Y1, bei P2 durch X2 und Y2. (Die Größen werden hier in Anlehnung an die Schreibweise innerhalb von Programmen groß geschrieben, in der Mathematik dagegen oft klein mit Indizes.)

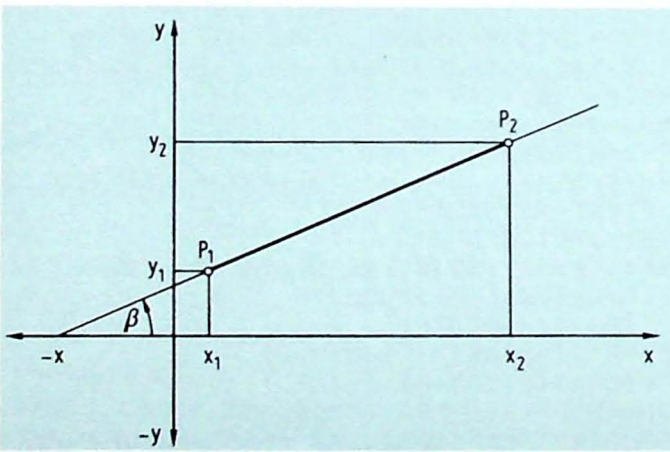


Abb. 5.4.5.1
Die Lage einer Gerade läßt sich durch zwei Punkte angeben

Zunächst zur Begriffsbestimmung. Eine gerade Linie zwischen zwei Punkten bezeichnet man streng als Strecke. Denkt man sich diese Linie in beide Richtungen bis in die Unendlichkeit fortgesetzt, spricht man dagegen von einer Geraden. Bei einer Geraden bestimmen zwei Punkte lediglich ihre Lage im Koordinatensystem, bei einer Strecke jedoch auch deren Länge.

Um eine Linie zeichnen zu können, muß der Computer in der Lage sein, für jeden X-Wert (horizontale Koordinate) den zugehörigen Y-Wert (vertikale Lage eines Punktes auf der Linie) zu berechnen. Hierzu gibt es eine einfache Formel:

$$Y = A*X + B$$

A ist hierbei ein Maß für die „Steigung" der Gerade, B ist die Stelle auf der y-Achse, durch die die Gerade verläuft. Wenn β (griech. Buchstabe Beta) der Winkel zwischen der Geraden und der x-Achse ist, so gilt für A in der obigen Gleichung auch:

$$A = TAN(\beta)$$

Sind zwei Punkte P1 und P2 der Geraden gegeben, so läßt sich A auch aus deren Koordinaten errechnen:

$$A = (Y2-Y1) / (X2-X1)$$

Der Geradenverlauf läßt sich mit einer etwas abgewandelten Formel ebenfalls so darstellen, daß bei zwei gegebenen Punkten nur noch bekannte Größen vorkommen:

$$Y = A*(X-X1)+Y1$$

Wenn man die beiden letztgenannten Formeln nun einsetzen möchte, um tatsächlich eine Linie auf den Bildschirm zu bringen, muß man noch wissen, daß auf vielen Computern im Gegensatz zur Mathematik der Nullpunkt (Ursprung) des Koordinatensystems nicht etwa links unten, sondern links oben auf dem Bildschirm ist. Größere Y-Werte entsprechen also tiefer liegenden Bildschirm-Positionen *(Abb. 5.4.5.2)*.

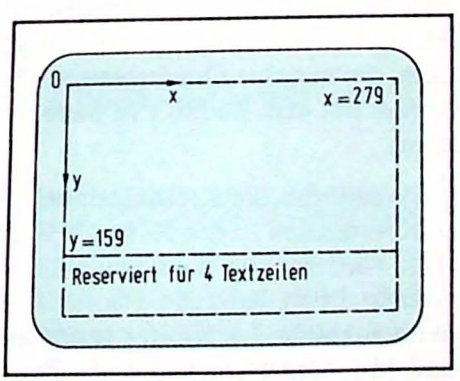


Abb. 5.4.5.2
Bei den meisten Computern ist die Richtung der Y-Achse gegenüber der Mathematik umgekehrt. Hier das Bildschirm-Format des Apple-II bei hochauflösender Grafik

```
5REM ZEICHNEN VON LINIEN
10HGR:HOME:VTAB22:HCOLOR=3:HIMEM:8192
20INPUT"X1,Y1,X2,Y2: ";X1,Y1,X2,Y2
30A=(Y2-Y1)/(X2-X1)
40FORX=X1TOX2:Y=A*(X-X1)+Y1
50YS=159-Y:HPLOTX,YS:NEXT:GOTO20
```

Abb. 5.4.5.3
Zeichnen einer Linie durch geeignetes Setzen von Punkten auf dem Schirm

Abb. 5.4.5.3 zeigt ein kurzes, in Applesoft-Basic geschriebenes Programm, das das Zeichen von Linien zwischen zwei Punkten mit den Koordinaten X1, Y1 und X2, Y2 gestattet. (Eigentlich wäre das beim Apple nicht nötig, da der Interpreter bereits Anweisungen zum Zeichnen ganzer Linien besitzt; hier geht es nur um die Demonstration des nötigen Algorithmus.) Zeile 30 errechnet die Steigung, und die erste Anweisung in Zeile 50 dreht die Lage der Y-Achse so, daß sie mathematisch korrekt von unten nach oben verläuft. Beim Apple-II hat man nach einer HGR-Anweisung die Möglichkeit, 278 Punkte in x- und 160 in y-Richtung darzustellen (0...279 und 0...159), was bei der Eingabe der Punkt-Koordinaten berücksichtigt werden muß.

Wenn man Linien innerhalb eines Maschinenprogramms zeichnen möchte, so eignet sich meist die sogenannte Punkt-Richtungs-Form der Geradengleichung besser, die die relative Position des nächsten Punktes vom gerade gezeichneten angibt:

$$Y-Y1 = A*(X-X1)$$

Vorteilhaft ist hier vor allem, daß nur eine geringe Zahl von Rechenoperationen zum Zeichnen des nächsten Punktes mit den Koordinaten X1, Y1 nötig ist.

Kreise

Während sich Linien mit sehr hoher Geschwindigkeit zeichnen lassen, sind für Kreise komplexere Rechenoperationen nötig (es sei denn, man verwendet eine Tabelle der nötigen Y-Werte). Die in der Mathematik manchmal vorgeschlagene Methode mit Sinus und Cosinus sei hier gar nicht erst erwähnt, da sie in der Praxis aus Geschwindigkeitsgründen kaum eingesetzt wird.

Abb. 5.4.5.4 zeigt eine Methode, wie man durch Zerlegen des Kreises in zwei Hälften die auf ihm liegenden Punkte in einer einfachen Funktion ausdrücken kann, die nur eine Quadratwurzel-Berechnung nötig macht. Der Mittelpunkt des Kreises mit dem Radius r ist hierbei zunächst der Ursprung des Koordinaten-Systems.

Da natürlich der Kreismittelpunkt frei wählbar sein muß, ist es erforderlich, seine Koordinaten zu den X- und Y-Werten der Kreisformel zu addieren. *Abb. 5.4.5.5* zeigt, wiederum im Basic-Dialekt des Apple-II, ein Beispielprogramm hierfür. Innerhalb der FOR-NEXT-Schleife wird abwechselnd ein Punkt der oberen und der unteren Hälfte des Kreises gezeichnet, dessen Mittelpunkt die Koordinaten MX und MY besitzt *(Abb. 5.4.5.6)*. Bei entsprechender Auslegung der Routine und

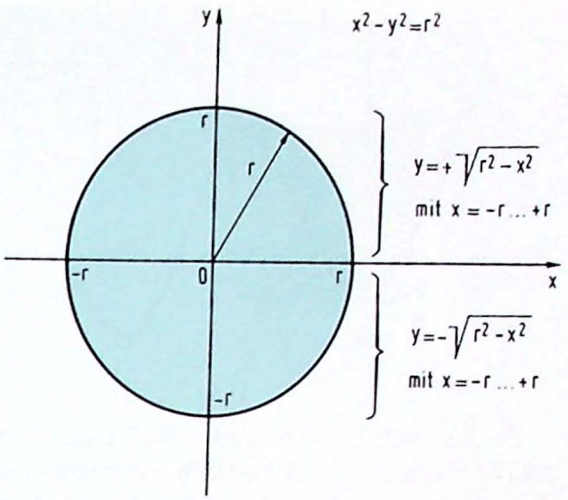


Abb. 5.4.5.4
Ein Kreis läßt sich mit wenigen Rechenoperationen zeichnen

Abb. 5.4.5.5
Applesoft-Programm zum Zeichnen eines Kreises an beliebiger Stelle auf dem Bildschirm und mit beliebigem Radius

```
5REM ZEICHNEN EINES KREISES
10HGR:HOME:VTAB22:HCOLOR=3:HIMEM:8192
20DEFFNK(X)=SQR(R*R-X*X)
30INPUT"X,Y,R: ";MX,MY,R
40FORX=-RTOR
50Y=MY+FNK(X):HPLOTX+MX,Y:REM OBERE HAELFTE
60Y=MY-FNK(X):HPLOTX+MX,Y:REM UNTERE H.
70NEXT:HPLOTMX,MY:REM MITTELPUNKT
```

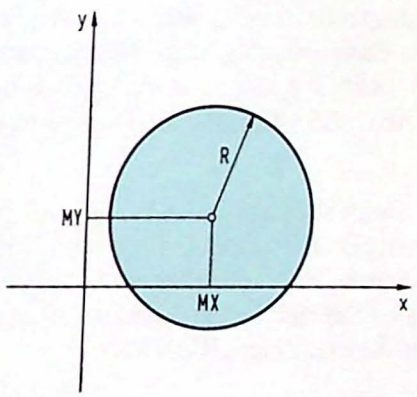


Abb. 5.4.5.6
Das Programm in Abb. 5.4.5.5 addiert zu allen Koordinaten des Kreisumfangs jene des Mittelpunktes

Berücksichtigung aller Symmetrien des Kreises könnten sogar jeweils acht Punkte auf einmal gezeichnet werden.

Zeichnen mathematischer Funktionen

Eine häufige Anwendung der Grafik-Möglichkeiten eines Computers ist die Darstellung des Verlaufs mathematischer Funktionen in anschaulicher Form. Das folgende Programmpaar zeigt, wiederum am Beispiel des Apple-II, wie so etwas benutzerfreundlich realisiert werden kann.

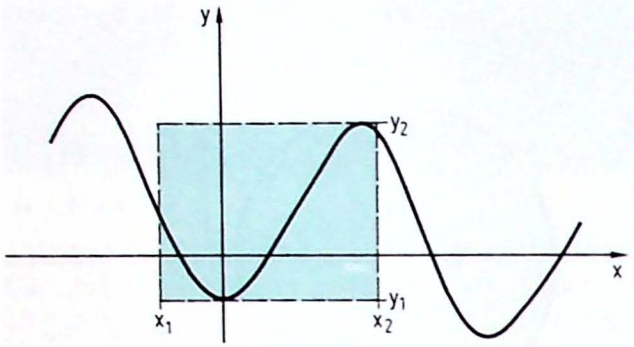


Abb. 5.4.5.7
Von jeder Funktion kann stets nur ein Ausschnitt auf dem Bildschirm dargestellt werden

Da der Bildschirm nicht unendlich groß ist, mathematische Funktionen dagegen meist beliebige X-Werte zulassen, kann man natürlich nur einen kleinen Ausschnitt des Verlaufs der Funktion darstellen *(Abb. 5.4.5.7)*. Um dem Benutzer langes Überlegen zu ersparen, stellt das Basic-Programm aus *Abb. 5.4.5.8* in einem Probe-Durchlauf selbst fest, welcher Y-Bereich hierbei zur Darstellung nötig ist, und spreizt diesen auf die gesamte Bildschirmhöhe, um den Schirm möglichst gut auszunutzen und so eine optimale Auflösung sicherzustellen.

Das bei Programmen für diesen Zweck meist am schwierigsten zu lösende Problem ist die Eingabe der Funktion bei laufendem Programm durch den Benutzer. Andernfalls wäre es nämlich nötig, daß der Benutzer die Funktion als neue Programmzeile eingibt, also selbst das Programm ändert. Um das zu vermeiden, lädt das Basic-Programm ein Hilfsprogramm nach, das in 6502-Assembler geschrieben ist *(Abb. 5.4.5.9)*. Es kann nach Eingabe vom Monitor aus mit BSAVE MEM-EXEC, A$300, L$40 auf Disk gespeichert werden.

Beim Start des Basic-Programms gibt der Benutzer die zu untersuchende Funktion per INPUT-Befehl in eine String-Variable ein, die dann als Bytefolge ab Adresse 1901 (dezimal 6400, ziemlich frei wählbar) abgelegt wird, allerdings ergänzt mit der Zeilennummer 910 und mit der durch ein Return-Zeichen getrennten Zeichenfolge „RUN200".

Das mit CALL 768 aufgerufene Maschinenprogramm MEM-EXEC (es muß unter diesem Namen auf Diskette stehen) tut nichts weiter, als die Tastatur-Eingabe vorübergehend auf die ab hex 1901 stehende Zeichenfolge umzuschalten. Dadurch wird die Funktion als neue Zeile 910 in das laufende (!) Basic-Programm übernommen. Die Anweisung RUN 200 sorgt anschließend dafür, daß der Benutzer nun den gewünschten X-Koordinaten-Bereich eingeben kann. Nach dem Probedurchlauf zum Ermitteln des Y-Darstellungsbereichs wird die Funktion auf dem Bildschirm angezeigt. Auf die Darstellung des Koordinaten-Kreuzes wird verzichtet, weil es u. U. ohnehin außerhalb des Bildschirms liegen würde.

```
10REM FUNKTIONSDARSTELLUNG
20TEXT:HOME:HIMEM:6400
30PRINT"FUNKTIONSDARSTELLUNG":PRINT
40IFPEEK(768)=162ANDPEEK(827)=3THEN60
50PRINTCHR$(4)"BLOAD MEM-EXEC"
60CR$=CHR$(13)
99:
100REM ZEILE 910 M.FUNKTION ERZEUGEN
110INPUT"FUNKTION Y(X): Y=";F$
120F$="910Y="+F$+CR$+"RUN200"+CR$
130FORI=1TOLEN(F$):X=ASC(MID$(F$,I,1))
140POKE6400+I,X+128:NEXT:POKE6400+I,0
150CALL768:END
199:
200REM EINGABE DES X-BEREICHS
210INPUT"XMIN,XMAX:";X1,X2:ONERRGOTO500
220IFX1>=X2THENPRINT"FEHLER!":GOTO210
230DX=(X2-X1)/278
239:
240REM Y-BEREICH AUSPROBIEREN
250Y1=1E30:Y2=-Y1:PRINT:PRINT"MOMENT!"
260FORX=X1TOX2STEPDX:GOSUB900
270IFY1>YTHENY1=Y
280IFY2<YTHENY2=Y
290NEXT:PRINT:IFY2>Y1THEN300
295PRINT"Y IST KONSTANT "Y"!":GOTO400
299:
300REM FUNKTION ZEICHNEN
310HOME:HGR:VTAB21:HCOLOR=3
320PRINT"XMIN="X1,"YMIN="Y1
330PRINT"XMAX="X2,"YMAX="Y2
340I=0:FORX=X1TOX2STEPDX:GOSUB900
350HPLOTI,160*(1-(Y-Y1)/(Y2-Y1))
360I=I+1:NEXT
390:
400REM ABFRAGE NEUEINGABE/ENDE
410VTAB24:PRINT"FUNKTION/KOORD/ENDE? ";
420GETK$:IFK$="F"THENRUN
430IFK$="K"THENHOME:VTAB21:GOTO200
440IFK$<>"E"THEN420
450HOME:TEXT:HIMEM:38400:END
499:
500REM FEHLERBEHANDLUNGS-ROUTINE
510HOME:VTAB21
520PRINT"FEHLER BEI X=";X:GOTO400
899:
900REM ZEILE 910=FUNKTION Y(X)
910REM WIRD HIER VOM PGM.ERZEUGT!
920RETURN
```

Abb. 5.4.5.8 Applesoft-Programm zum Zeichnen einer beliebigen Funktion

```
0300             1          ORG $300
0300             2  ;MEM-ECEC:
0300             3  ;BEFEHLE AB TXT NACH
0300             4  ;CALL 768 AUSFUEHREN
0300             5  ;(C) FE./16.1.85
0300             6  SAVX    EPZ $FD
0300             7  PNT     EPZ $FE
0300             8  OUTV    EPZ $36
0300             9  KBV     EPZ $38
0300            10  TXT     EQU $1901
0300            11  ;VEKTOREN RETTEN
0300            12  ;UND AENDERN
0300 A203       13  INIT    LDX #3
0302 B536       14  INIT1   LDA OUTV,X
0304 9D3C03     15          STA TEMP,X
0307 BD3803     16          LDA TAB,X
030A 9536       17          STA OUTV,X
030C CA         18          DEX
030D 10F3       19          BPL INIT1
030F            20  ;ZEIGER AUF TXT
030F A901       21          LDA #TXT
0311 85FE       22          STA PNT
0313 A919       23          LDA /TXT
0315 85FF       24          STA PNT+1
0317 60         25          RTS
0318            26  ;ZEICHEN HOLEN
0318 A000       27  GET     LDY #0
031A B1FE       28          LDA (PNT),Y
031C 48         29          PHA
031D C8         30          INY
031E B1FE       31          LDA (PNT),Y
0320 300E       32          BMI NEND
0322            33  ;VEKTOREN RUECKS.
0322 86FD       34          STX SAVX
0324 A203       35          LDX #3
0326 BD3C03     36  RST     LDA TEMP,X
0329 9536       37          STA OUTV,X
032B CA         38          DEX
032C 10F8       39          BPL RST
032E A6FD       40          LDX SAVX
0330 68         41  NEND    PLA
0331            42  ;ZEIGER ERHOEHEN
0331 E6FE       43  GET1    INC PNT
0333 D002       44          BNE GET2
0335 E6FF       45          INC PNT+1
0337 60         46  GET2    RTS
0338            47  ;VEKTOREN-TAB.
0338 3703       48  TAB     ADR GET2
033A 1803       49          ADR GET
0340            50  TEMP    DFS 4
                51          END
```

Abb. 5.4.5.9 Dieses Hilfsprogramm sorgt dafür, daß der Rechner die Zeichen nicht von der Tastatur, sondern vorübergehend aus dem RAM holt

6 Adressen

6.1 Hersteller von Computern und Peripherie

AAA (aaa); Habsburger Str. 134; 7800 Freiburg
AC-Copy; Kurbrunnenstr. 30; 5100 Aachen
Abakus; Nohlstr. 29; 4200 Oberhausen 1
Acorn; Anzinger Str. 1; 8000 München 80
Adcomp; Olgastr. 15; 8000 München 19
Altos; Rudolf-Diesel-Str. 1; 8032 Gräfelfing
Apple; Ingolstädter Str. 20; 8000 München 45
Astronic; Winzererstr. 47 d; 8000 München 40
Atari; Bebelallee 10; 2000 Hamburg 60

BASF; Karl-Bosch-Str. 38; 6700 Ludwigshafen
BMC; Roßmarkt 15; 6000 Frankfurt
Bierbrauer + Nagel; Breitwiesenstr. 5; 7000 Stuttgart 80
Binder; Mönchsweilerstr. 1; 7730 VS-Villingen
Bitronic; Einsteinstr. 127; 8000 München 80
Bremer Computer Shop; Hastedter Heerstr. 391; 2800 Bremen
Brosius + Köhler; Unterdörnen 71; 5600 Wuppertal 2
Brother International GmbH; Postfach 1320; 6368 Bad Vilbel

CC Computer Studio; Elisabethstr. 5; 4600 Dortmund 1
CSBV; Raiffeisenstr. 13; 8068 Pfaffenhofen/Ilm 1
CTH; Bismarckstr. 5/18; 6360 Friedberg
CTM; Max-Stromeyer-Str. 37; 7750 Konstanz
Canon; Postfach; 8033 Martinsried
Centronics; Lyoner Str. 44; 6000 Frankfurt 71
Commodore; Lyoner Str. 38; 6000 Frankfurt 71
Compaq; Arabellastr. 30; 8000 München 81
Computer Accessoires; Jägerweg 10; 8012 Ottobrunn
Computershop (MSB); Mangoldstr. 10; 7778 Markdorf

DS (Data-Service); Königstr. 20—22; 6750 Kaiserslautern
Damm + Johanning; Sudbrackstr. 46—48; 4800 Bielefeld 1
Data Becker; Merowingerstr. 30; 4000 Düsseldorf
Data General; Stuttgarter Str. 10; 6236 Eschborn
Deutsche Eurotech GmbH; Grünstr. 32; 4005 Meerbusch 1

Digital Equipment; Freischützstr. 91; 8000 München 81
Digitronic; Am Kamp 17; 2081 Holm
Durango Systems GmbH; Breslauer Str. 60; 8500 Nürnberg

EES; Pelkovenstr. 49; 8000 München 50
EKF; Am Pilsholz 4; 4700 Hamm 1
Eckman-Ecotronic; Wanheimer Str. 39; 4000 Düsseldorf 30
Elbit Computer GmbH; Postfach 520129; 5000 Köln
Elphotec; Schießgartenstr. 7; 6500 Mainz
Eltec; Postfach 1847; 6500 Mainz
Epson; Zülpicher Str. 6; 4000 Düsseldorf 11
Erma-Chemie-Computer; Schulstr. 12; 5451 Strassenhaus
Eschenbrenner + Sauer; Alte Weinsteige 40; 7000 Stuttgart 1
Exidy; C-Data; Am Südpark 7 b; 6092 Kelsterbach

Facit/Ericsson; Niederheiderstr. 3; 4000 Düsseldorf 13
Feltron; Auf dem Schellerod 22; 5210 Troisdorf
Fey; Horemansstr. 28; 8000 München 19
Force; Freischützstr. 92; 8000 München 81
Fortune System GmbH; Am Roßmarkt 15; 6000 Frankfurt
Fujitsu; Postfach 710225; 6000 Frankfurt 71

GC (General Consult); Gervinusstr. 4; 1000 Berlin
GC (General Consult); Landwehrstr. 61; 3000 Hannover 81
GDE; Nürtinger Str. 41; 7024 Filderstedt 1
GMC; Ostdamm 30; 4408 Dülmen
GWK; Asternstr. 2; 5120 Herzogenrath
Gemini; Briennerstr. 56; 8000 München 2
General Corporation; Wolfstr. 20; 4000 Düsseldorf 11
Genisco; Stolberger Str. 90; 5000 Köln 41
Gerb; Rödernallee 174—176; 1000 Berlin 51
Güssow; Stolper Str. 2; 2000 Hamburg 73

Heath; Robert-Bosch-Str. 32; 6072 Dreieich
Heckler + Koch; Breslauer Weg 37; 8192 Geretsried
Hewlett-Packard (HP); Berner Str. 117; 6000 Frankfurt 56
Honeywell-Bull; Hohenstaufenring 62; 5000 Köln 1
Hüllen; Heinrich-Späth-Str. 12—14; 4019 Monheim

IBM; Postfach 800880; 7000 Stuttgart 80
ICL Deutschland; Marienstr. 10; 8500 Nürnberg 1
ISE-Computer; Lothstr. 19; 8000 München 40

ISI; Unterhachinger Str. 28; 8012 Ottobrunn
ITT; Postfach 1720; 7530 Pforzheim
Implementa GmbH; Niedersachsenstr. 25; 3012 Langenhagen b. Hannover
Info-Process; An der Ries 8; 7801 Hartheim
Info-Process; Mitteltal 27; 7801 Stegen 2
Interautomation; Marburger Str. 10; 1000 Berlin 10
Intertec Data Systems; Postfach 68; 8070 Ingolstadt
Itoh; Königsallee 21; 4000 Düsseldorf

KBJ; Odenthalerstr. 136; 5060 Bergisch-Gladbach 2
Kanis; Lindenberg 113; 8134 Pöcking
Karp; Großmoorring 7; 2100 Hamburg 90
Kaypro; Roßmarkt 15; 6000 Frankfurt
Kienzle Apparate GmbH; Postfach 1640; 7730 VS-Villingen
Kontron; Freisinger Str. 21; 8057 Eching

LSI Electronic; Orionstr. 2; 8044 Unterschleißheim
Landolf; Wingertstr. 112; 6457 Maintal 1
Lange Elektronik; Fürholznerstr. 1; 8057 Eching
Leuwico; 8631 Wiesenfeld-Coburg
Linde; Neue Str. 18; 7170 Schwäbisch Hall
Logabax Deutschland GmbH; Sophienstr. 26; 6000 Frankfurt/Main 90
Luxor; Dohlenweg 1; 4156 Willich 2

MAI; Hahnstr. 31—35; 6000 Frankfurt/Main 71
MCC (Christ); Rathausstr. 4; 2300 Kiel 1
MCM; Lyoner Str. 44; 6000 Frankfurt 71
MDS Deutschland GmbH; Oskar-Jäger-Str. 175; 5000 Köln 30
MM (Michael Matrai); Europaplatz 20; 7000 Stuttgart 80
MMC-Mertes Mikrocomputer; Hahnenstr. 12 a; 5020 Hürth 5
MVB GmbH; Brüder-Grimm-Str. 5; 6408 Ebersburg-Weyhers
Magirus Datentechnik GmbH; Dieselstr. 24; 7022 Leinfelden-Echterd
Mannesmann-Tally; Postfach 2969; 7900 Ulm
Marubeni; Immermannstr. 43—45; 4000 Düsseldorf 1
Meilhaus; Fischerstr. 2; 8039 Puchheim
Melchers & Co.; Schlachte 39/40; 2800 Bremen 1
Micro Logic GmbH; Dingolfingerstr. 2; 8000 München 80
Micro; Alsfelder Str. 7; 6100 Darmstadt
Micro; Dreieichstr. 59; 6000 Frankfurt 70
Microdata GmbH; Postfach 2015; 7850 Lörrach
Micropoint; Werder Str. 18 a; 8500 Nürnberg 20
Mitsui; Königsallee 92 a; 4000 Düsseldorf
Motorola; Münchner Str. 18; 8043 Unterföhring
Münch; Brinkstr. 43; 2842 Lohne
Mylius; Herzogstr. 79; 8000 München 40

NCR GmbH; Ulmer Str. 160; 8900 Augsburg
NEC Home Electronics; Wiesenstr. 148; 4040 Neuß
National Panasonic; Jungfernsteig 40; 2000 Hamburg 36
Nixdorf; Fürstenallee 7; 4790 Paderborn
Nokia Data GmbH; Petersbrunnerstr. 8 a; 8130 Starnberg
Nordex Datentechnik; Hermannstr. 46; 2000 Hamburg 1
Norsk-Data Dietz; Abraham-Lincoln-Str. 30; 6200 Wiesbaden
Northstar; Arabellastr. 5/133c; 8000 München 21

OCS Olga Ölzner; Kaiser-Wilhelm-Ring 12; 5000 Köln 1
Oki (OKI); Emanuel-Leutze-Str. 8; 4000 Düsseldorf 11
Olivetti; Kaiser-Friedrich-Promenade 89; 6380 Bad Homburg 1
Olympia; Postfach 960; 2940 Wilhelmshaven
Onyx Computer Corp.; Auf dem Berg 2; 6230 Frankfurt 80
Osborne; Dingolfinger Str. 6; 8000 München 80

PHS; Davenstedter Str. 8; 3000 Hannover 91
Panasonic (Matsushita); Winsbergring 15; 2000 Hamburg 54
Percom Computerperipherie; Eisenacherstr. 10; 8000 München 40
Philips Datasystems GmbH; Weidenauerstr. 211—213; 5900 Siegen 21
Philips; Postfach 101420; 2000 Hamburg 1
Plastotex; Am Steinacher Kreuz 22; 8500 Nürnberg 90
Printec DTV; Daimlerstr. 14; 6452 Hainburg
Proteus GmbH; Heidelberger Str. 11; 7514 Leopoldshafen
Proxa; Aachener Str. 29; 5000 Köln 1

Qume; Eichelstr. 31; 4000 Düsseldorf 12

RC Computer; Lyoner Str. 44—48; 6000 Frankfurt/Main 71
Rair Computer GmbH; Clemensstr. 5—7; 5000 Köln 1
Rank Xerox GmbH; Emanuel-Leutze-Str. 20; 4000 Düsseldorf
Redactron Deutschland Vertriebs GmbH; Vor der Lauch 9; 7000 Stuttgart 80
Rockwell; Fraunhoferstr. 11; 8033 Martinsried
Rohner-EDV; Allgäuer Str. 77; 8950 Kaufbeuren

SGS-Ates; Haidling 17; 8018 Grafing
SKS; Maybachstr. 10; 7500 Karlsruhe 41
Salota GmbH; Raabstr. 24; 8500 Nürnberg
Sanyo Büro Elektronik Europa; Truderingerstr. 13; 8000 München 80
Sanyo Video Systeme; Lange Reihe 29; 2000 Hamburg 1
Schneider-Werke; Silvastr. 1; 8930 Türkheim
Sharp; Sonninstr. 3; 2000 Hamburg 1
Shugart; Benzstr. 28; 8039 Puchheim
Siebeneicher; Postfach 9; 3044 Neuenkirchen
Siemens-Infoservice; Postfach 156; 8501 Fürth

Siemens; Postfach 103; 8000 München 1
Sinclair Research Ltd.; Ottostr. 28; 8012 Ottobrunn
Sinclair; Erlenweg 3; 8028 Taufkirchen
Solarim; Roritzer Str. 37; 8500 Nürnberg 91
Sony Deutschland GmbH; Hugo-Eckener-Str. 20; 5000 Köln 30
Spezial Electronic; Hermann-Lingg-Str. 16; 8000 München 2
Spima-Computer; Turbinenstr. 4; 6800 Mannheim 31
Standard Elektrik Lorenz AG; Östliche 132; 7530 Pforzheim
Sträßle Datentechnik GmbH; Herdweg 29—31; 7000 Stuttgart
System-Kontakt GmbH; Siemensstr. 5; 7107 Bad Friedrichshall

TCV-Strötmann; Nordfelder Reihe 27; 3000 Hannover 1
TEC Electronik Deutschland GmbH; Christinenstr. 15; 4030 Ratingen 1
Tandberg Data; Feldstr. 81; 4600 Dortmund 1
Tandy; Christinenstr. 11; 4030 Ratingen 1
Tandy; Luisenstr. 98; 4000 Düsseldorf 1
Taylorix; Zazenhäuserstr. 106; 7000 Stuttgart 40
Teach; Siemensstr. 22; 7257 Ditzingen 1
Tektronix; Sedanstr. 13; 5000 Köln 1
Tewidata; Romanstr. 35—37; 8000 München 19
Texas Instruments; Haggertystr. 1; 8050 Freising
Textcom; Glockengießer Wall 26; 2000 Hamburg 1
Thomson-CSF; Bogenstr. 51; 5400 Koblenz
Thun; Johannisstr. 7; 2350 Neumünster
Time-soft (time-soft-EDV); Sophienstr. 32; 7000 Stuttgart 1
Toshiba; Hammer Landstr. 115; 4040 Neuß
Trenddata; Am Marstall 20—22; 3000 Hannover 1
Triumph-Adler AG Vertrieb Mikrocomputer; Wischelstr. 71; 8500 Nürnberg
Triumph-Adler; Fürther Str. 212; 8500 Nürnberg
Trommeschläger; Postfach 2105; 5205 St. Augustin 2

Unitronic; Münsterstr. 338; 4000 Düsseldorf 30

Victor/Sirius; Sontraer Str. 18; 6000 Frankfurt 61
Video Data Systems GmbH; Beethovenstr. 26; 6455 Erlensee
Vobis; Viktoriastr. 74; 5100 Aachen
Vollrath; Kirchstr. 28; 4650 Gelsenkirchen

Wang Deutschland GmbH; Lyoner Str. 26; 6000 Frankfurt/Main
Watanabe; Arzbergerstr. 10; 8036 Hersching
Weber; Emil-von-Behring-Str. 6; 2800 Bremen
Wiesemann; Winchenbachstr. 3a; 5600 Wuppertal

Ziegler Instruments; An der Waldesruh 17; 4050 Mönchengladbach
Zilog; Eschenstr. 8; 8028 Taufkirchen

6.2 Hersteller von Bauelementen und Baugruppen

AFU (afu); Steinstr. 9; 5778 Meschede
AMD; Rosenheimer Str. 143 a; 8000 München 80
ASE-Soft; Hahnweiler Str. 11; 6761 Steinbach
Apesoft; A-2431 Enzersdorf; Österreich
Astronic; Winzererstr. 47 d; 8000 München 40
Atlantik; Fraunhofer Str. 11; 8033 Martinsried

Bartscher; Westring 84; 3440 Eschwege
Beisch; Josef-von-Görres-Str. 49; 5100 Aachen
Bekatron; Postfach 1125 P; 8907 Thannhausen
Berg; Wasserwegstr. 92; 4500 Osnabrück
Berger; Obermatten 11; 7803 Gundelfingen
Binder; Postfach 1226; 7730 Villingen-Schwenningen
Bingger; Luitpoldstr. 14; 8034 Germering
Bitronic; Einsteinstr. 227; 8000 München 80
Blender & Springer; Hauptstr. 38; 7480 Sigmaringen
Braunschmidt; Inzersdorfer Str. 119; A-1100 Wien

CE Computer-Systeme; Marktstr. 8; 4150 Krefeld
Carrot's; Postfach 1165; 4690 Herne 1
Computer Elektronik G. Krause; Zum Römergrund 19; 6501 Wörrstedt
Corvallis-Team; Postfach 1125; 6882 Friedrichsdorf

DEC (Digital Equipment); Freischützstr. 91; 8000 München 81
Digitronic; Am Kamp 17; 2081 Holm
Dittrich; Pflugweg 3; 7570 Baden-Baden 24

E-Lab/Hofmann; Grombacher Str. 21; 6927 Bad Rappenau 5
EKF; Am Pilsholz 4; 4700 Hamm 1
Ebeco (EBECO); S. König; St.-Anno-Str. 6; 5000 Köln 90
Eckelmann; Alte Schmelze 21; 6200 Wiesbaden
Eichweber; Schützenstr. 75; 2000 Hamburg 50
Elcoprint; Aidenbacher Str. 25; 8358 Vilshofen
Elektronikladen; Wilhelm-Mellies-Str. 88; 4930 Detmold 18
Elmetic; Alte Schmelze 21; 6200 Wiesbaden
Elsa; Haketweg 20; 2190 Stolberg-Büsbach
Eltec; Postfach 1847; 6500 Main

Fairchild; Daimlerstr. 15; 8046 Garching-Hochbrück
Feise; Alte Zeche 2; 3013 Barsinghausen
Feltron; Postfach 1169; 5210 Troisdorf-Spich
Force; Freischützstr. 92; 8000 München 81
Frank; Matthiasstr. 3; 8500 Nürnberg 40

GTE Microcircuits; Montenstr. 11; 8000 München 19
GWK; Asternstr. 2; 5120 Herzogenrath
Geotek; Blücherstr. 32; 1000 Berlin 61
Görlitz; Postfach 852; 5400 Koblenz
Graf (GES); Magnusstr. 13; 8960 Kempten

HW; Eimsb. Chaussee 79; 2000 Hamburg 19
Havekost; Ringbahnstr. 65; 1000 Berlin 42
Heinz; Kaiserstr. 57; 4600 Dortmund
Hertrich; Im Möllenkamp 26; 4830 Gütersloh 1
Hewlett-Packard; Berner Str. 117; 6000 Frankfurt 56

IS-Systems; Hauptstr. 27; 5340 Bad Honnef
ITT-Intermetall; Postfach 840; 7800 Freiburg
Impec; Schaffhauser Str. 113; 7400 Tübingen
Intel; Seidlstr. 27; 8000 München 2
Intersil; Bavariaring 38; 8000 München 15
Intex; Perchtinger Str. 3; 8000 München 70

Janich + Klass; Neumarktstr. 46; 5600 Wuppertal
Jann; Glimmerweg 22; 1000 Berlin 47

Kaeser; Reifenstülstr. 8; 8000 München 5
Kanis; Lindenberg 113; 8134 Pöcking
Köbbing; Römerstr. 3; 4600 Dortmund 1
Kontron; Breslauer Str. 2; 8057 Eching

LCS Logosoft; Bültestr. 170; 4972 Löhne 3
Lakosa; Ginsterweg 15; 4790 Paderborn
Launee; Nymphenburger Str. 147; 8000 München 19
Leunig; Pinner Str. 7; 5206 Neunkirchen-Seelscheid 1

MEK; Gneisenaustr. 27; 2300 Kiel
Manger; Postfach 1127; 7800 Freiburg
Marflow; Brüderstr. 2; 3000 Hannover 1
Metronik; Kapellenstr. 9; 8025 Unterhaching
Microphase GmbH; Edelsberger Str. 8; 8000 München 21
Mitsubishi; Brandenburger Str. 40; 4030 Ratingen
Mostek; Freischützstr. 92; 8000 München 81
Motorola; Münchner Str. 18; 8043 Unterföhring

NEC; Harkortstr. 15; 4000 Düsseldorf
National Semiconductor; Industriestr. 10; 8080 Fürstenfeldbruck
Neucom; Hangweg 4; 8893 Hilgertshausen-Tandern
Nordcom; Düppelstr. 71; 2300 Kiel

OCS; Schillerstr. 7; 4930 Detmold
Oettle + Reichler; Adalbert-Stifter-Str. 40; 8902 Neusäß

Plessey; Altheimer Eck 10; 8000 München 2

R + R (r + r); Adlerstr. 55; 6900 Heidelberg 1
RB-Elektronik-Vertrieb; Bouraueler Str. 13; 5208 Eitorf
RCA; Justus-von-Liebig-Ring 10; 2085 Quickborn
Ratev; Postfach 1601; 4030 Ratingen 1
Rockwell; Fraunhoferstr. 11; 8033 Martinsried
Roos; Kleiner Markt 7; 4190 Kleve

SGS; Haidling 17; 8018 Grafing
Scantec; Landshuter Allee 79; 8000 München 19
Siemens; Postfach 103; 8000 München 1
Solarim; Roritzerstr. 28; 8500 Nürnberg 91
Statistics + Software W. Fassbender; Postfach 132; 1000 Berlin 42
Stöhr; Friedensstr. 22; 5190 Stolberg
Synertek; Oskar-von-Miller-Ring 29; 8000 München 2
Systec; Postfach 247; 4404 Telgte
System-Kontakt; Siemensstr. 5; 7107 Bad Friedrichshall

Tecsys (TecSys); Arabellastr. 13; 8000 München 81
Texas Instruments; Haggertystr. 1; 8050 Freising
Thaler; Magdeburger Str. 81; 4150 Krefeld
Thompson-CSF; Perchtinger Str. 3; 8000 München 70
Tiltz; Postfach 40; 8751 Leidersbach
Time + Soft (time + soft); Sophienstr. 32; 7000 Stuttgart 1

Unitronic; Münsterstr. 338; 4000 Düsseldorf 30

Valvo; Burchardstr. 19; 2000 Hamburg 1

Weiss; Am Wiesenhof 17; 2940 Wilhelmshaven
Weiss; Talstr. 136; 6905 Schriesheim
Winzen; Kölner Str. 67; 5090 Leverkusen
Wirth; Mühlstr. 25; 7064 Remshalden

Zilog; Eschenstr. 8; 8028 Taufkirchen

6.3 Hersteller von Akustikkopplern

Anderson-Jacobsen; Im Luchsfeld 5; 5060 Bergisch-Gladbach 1

Beate Vollrath Computer; Kirchstr. 17 + 28; 4650 Gelsenkirchen

CDI; Tauentzienstr. 1; 1000 Berlin 30
CTK; Langenbrück 20; 5060 Bergisch-Gladbach

Daum; Kagenhof 3; 8501 Veitsbronn

EDV-Kontor; Magdalenenstr. 26; 2000 Hamburg 13
Epson; Zülpicher Str. 6; 4000 Düsseldorf 11
Erbrecht; Lappenbergsallee 37; 2000 Hamburg 19
Eurotech; Grünstr. 32; 4005 Meerbusch

HIB; Äußere Bayreuther Str. 72; 8500 Nürnberg 21

Microphase GmbH; Edelsberger Str. 8; 8000 München 21
MSI Data; Horbellerstr. 10—14; 5000 Köln 40
Modular Technology; Postfach 270152; 4000 Düsseldorf 11

Olivetti; Lyoner Str. 34; 6000 Frankfurt 71
Oppermann; Harksheider Weg 10; 2085 Quickborn

R + R (r + r); Adlerstr. 55; 6900 Heidelberg

Software-Express; Hugo-Viehoff-Str. 84; 4000 Düsseldorf 30
Systec; Postfach 247; 4404 Telgte

Tandy; Christinenstr. 11; 4030 Ratingen 1
Telemation; Bismarckstr. 8; 6232 Bad Soden
Telemation; Liebigstr. 12; 8047 Karlsfeld

Wetronic Automation; Claude-Lorrain-Str. 27; 8000 München 90
Wörlein; Hindenburgstr. 37; 8501 Cadolzburg

6.4 Software-Hersteller

Abeln, Wolfgang; Adolphstr. 36; 5000 Köln 21

Bauer; Flockenfeld 117; 4200 Oberhausen 1
Berger; Obermatten 11; 7803 Gundelfingen
Bernd; Lambertusstr. 23; 5210 Troisdorf

Comidos Software; Krefelder Str. 289; 4100 Duisburg-Rheinhausen
Computer-Hüsli; A. Weyer; Postfach 860808; 8000 München 86
Comsoft (ComSoft); Fuggerstr. 4; 8901 Stadtbergen 2
Corvallis-Team; Postfach 1125; 6882 Friedrichsdorf

Data Becker; Merowingerstr. 30; 4000 Düsseldorf
Decker; Kalchreuther Str. 136; 8500 Nürnberg
Digital Research; Hansastr. 15; 8000 München 21

EDV-Formtext T. Hanitzsch; Am Heilborn 1; 3554 Lohra
Edition Duke Electronics; Postfach 1847; 6500 Mainz

Flesch; Schützenstr. 3; 7820 Titisee-Neustadt
Förster; Mauthstr. 1; 8253 Buchbach
Franzis-Software-Service; Postfach 370120; 8000 München 37

Heimsoeth; Fraunhoferstr. 13; 8000 München 5
Holtkötter; Albert-Schweitzer-Ring 9; 2000 Hamburg 70
Hüthig; Postfach 102869; 6900 Heidelberg

ISOS (Isos); Reutershagweg 23; 5100 Aachen

Joysoft; Humboldtstr. 4; 4000 Düsseldorf

Kraml; Degengasse 27/16; A-1160 Wien

MicroPhase; Nymphenburgerstr. 178; 8000 München 40
MicroPro (Micropro); Berg-am-Laim-Str. 127; 8000 München 80
Microsoft; Erdinger Landstr. 2; 8011 Aschheim-Dornach

Profisoft; Sutthauser Str. 50/52; 4500 Bad Friedrichshall

RVS; Postfach 55; 8055 Hallbergmoos
Röckrath; Noppiusstr. 19; 5100 Aachen

SM; Scherbaumstr. 33; 8000 München 83
Software 2000; Kaiserstr. 35; 8000 München 40
Statistics + Software; Postfach 132; 1000 Berlin 42

Teldec; Heussweg 25; 2000 Hamburg 19
Time-soft (time-soft-EDV); Sophienstr. 32; 7000 Stuttgart 1

Video-Magic (Videomagic); Sonnenstr. 9; 8000 München 2

Wilke (IDA); Wilhelmstr. 72; 5100 Aachen 1

Zabel; Kirschallee 12; 5300 Bonn 1

6.5 Institutionen und Verbände

Bundesamt für Wehrtechnik und Beschaffung; Am Rhein 2; 5400 Koblenz
Bundeskartellamt; Mehringdamm 129; 1000 Berlin 61

Deutsche Forschungsgemeinschaft; Kennedy-Allee 40; 5300 Bad Godesberg
Deutsches Institut für Normung, DIN; Burggrafenstr. 4; 1000 Berlin 30
Deutsches Patentamt; Zweibrückenstr. 12; 8000 München 2
Deutsches Patentamt, Zweigstelle Berlin; Gitschiner Str. 97—103

Fernmeldetechnisches Zentralamt, FTZ (DBP); Postfach 5000; 6100 Darmstadt

Normenausschuß Elektrotechnik; Stresemannallee 21; 6000 Frankfurt

Technischer Überwachungs-Verein; Kurfürstenstr. 56; 4300 Essen

Verband Deutscher Elektrotechniker, VDE; Stresemann-Allee 15; 6000 Frankfurt

Verein Deutscher Ingenieure, VDI; Graf-Recke-Str. 84; 4000 Düsseldorf

Zentralamt für Zulassungen im Fernmeldewesen, ZZF (DBP); 6600 Saarbrücken

Zentralverband der Elektrot. Industrie, ZVEI; Stresemannallee 19; 6000 Frankfurt

7 Quellenverzeichnis

Kapitel 1

Nührmann, D.: Werkbuch Elektronik. Franzis-Verlag, München.
Troitzsch, M.: Mikrocomputer-Schaltungstechnik. Franzis-Verlag, München.
Feichtinger, H.: Mit Computern steuern. Franzis-Verlag, München.
Feichtinger, H.: Mikrocomputer von A bis Z. Franzis-Verlag, München.
Feichtinger, H.: CMOS löst Probleme. mc 11/1984, S. 48.
Feichtinger, H.: Babylon's Datenverwirrung. mc 1/1981, S. 19.
Karsten, W.: Der Strichcode. mc 1/1981, S. 32.
Pelka, H.: Der Einchip-Mikrocomputer. Franzis-Verlag, München.
Stuhlmüller, P.: 16-Bit-Generation Z8000. Te-Wi-Verlag, München.
Das EMUF-Sonderheft. Herausgegeben von der Zeitschrift mc, 1982.
Mikrocomputer Schritt für Schritt. Sonderheft der Zeitschrift mc, 1984.

Kapitel 2

Klein, R. D.: Floppy-Disk-Aufzeichnungsverfahren. mc 7/1984, S. 38.
Das Modem-Sonderheft. Herausgegeben von der Zeitschrift mc, 1983.
Deutsche Bundespost (Hrsg.): Datex-P-Info-Broschüre.
Dietz, K.: Hier spricht der Computer. mc 5/1984, S. 42.
Jäger, R.: Das mc-Modem. mc 1/1985, S. 40.
Lesea, A.; Zaks, R.: Mikrocomputer-Interface-Techniken. Sybex, Düsseldorf.
Piotrowski, Dr. A.: IEC-Bus. Franzis-Verlag, München.
Plate, J.: Sprachausgabe für Tischcomputer. mc 11/1982, S. 44.
Datenkommunikation. Sonderheft der Zeitschrift Elektronik, 1983.

Kapitel 3

Plate, J.; Wittstock, P.: Pascal. Franzis-Verlag, München.
Leckebusch, J.: Was ist USCD-Pascal? mc 1/1982, S. 34.
Löschner, J. G.: Eigenarten von Turbo-Pascal. mc 1/1985, S. 90.
Busch, R.: Basic für Einsteiger. Franzis-Verlag, München.
Kernighan, R.: Programmieren in C. Hanser-Verlag, München.
Microsoft (Hrsg.): MBasic-Handbuch.
Zech, R.: Die Programmiersprache Forth. Franzis-Verlag, München.

Graves (Hrsg.): Forth for the Apple-II.
Kobenter, R.: Streifzug durch die Sprache C. mc 2/1985, S. 42.
Tonretzky, D.: Lisp. Harper & Row, New York.
Wirth, N.: Compilerbau. Teubner-Verlag, Stuttgart.
Wunderlich, F.: Erfolgreicher mit CBM arbeiten. Franzis-Verlag, München.

Kapitel 4

Klein, R. D.: CP/M, Betriebssystem für jedermann. mc 1/1983, S. 42.
Pol, B.: CP/M-86 oder MS-DOS oder was? mc 3/1984, S. 36.
Oettle, R.; Reichler, Th.: Der mc-CP/M-Plus-Computer. mc 9/1984, S. 86.
Olivetti (Hrsg.): MS-DOS-Handbuch zum M-24.
Plate, J.: Betriebssystem CP/M. Franzis-Verlag, München.
Müller, C. M.: Unix, ein Betriebssystem mit Zukunft. mc 1/1982, S. 24.
Rudolf, D. J.: Das ist Unix. mc 2/1985, S. 44.
Silvester, P. P.: The Unix System Guidebook. Springer-Verlag, Berlin.
Stanka, Z.; Lösch, S.: Unix. Te-Wi-Verlag, München.
Apple (Hrsg.): Das DOS-Handbuch.
Das Apple-Sonderheft. Herausgegeben von der Zeitschrift mc, 1984.

Kapitel 5

Bartsch, H. J.: Mathematische Formeln. Verlag Buch + Zeit, Köln.
Haugg, F.: Software-Engineering. Franzis-Verlag, München.
Kahlen, H.: Urheberrechtlicher Schutz für Software. mc 11/1984, S. 52.
Feichtinger, H.: Basic für Mikrocomputer. Franzis-Verlag, München.
Feichtinger, H.: Basic (nicht nur) für Anfänger. mc 1/1985, S. 79.
Domann, Dr. K.; Hera, Th.: Bestimmung von Bahnlängen. mc 1/1984, S. 64.
Plate, J.: Suchen und Sortieren in Basic und Pascal. mc 1/1981, S. 25.
Schulz, W.: Der gefädelte binäre Suchbaum. mc 4/1984, S. 138.
Zahmt, C.: Datenverarbeitungs-Verträge. CW-Publikationen, München.

Sachverzeichnis

A

A/D-Umsetzer 120, 268, 304
Ablenk|frequenz 127, 199
-spule 198
Absolut 67
ACIA 112
ADA 426
Adressen 597
-belegung 253, 310, 317, 319
-bereich 244
-bus 11, 12, 48, 179
Adressierungsarten 67
AIM-65 315
AIM-65-Format 232
Akku 36, 138
Akustikkoppler 278, 605
Algebra, Boolesche 151
Algol 426
Algorithmen 576
Alphanumerische Anzeige 195
ALT-Taste 337, 490
ALU 33
AM 125, 273
Amplitudenmodulation 273
Analogschalter 162
AND 152
Anode 147, 198
ANSI-Codes 492
Answer 279
Antennen-Eingang 126
Anwender-Software 541
Anzeigen 566
APL 426
Apple-DOS 3.3 517
– -II 309, 365, 517, 574
-soft-Basic 365
-writer 574
Arbitration 185
Arithmetik 347, 364, 585
Array 349, 438
ASCII 21
ASM 340
Assembler 323
Ätzen 175
Auffrischlogik 94
Ausgabeschaltung 100
AUTOEXEC.BAT 470

B

Backspace 22
BAM 532
Bandbreite 129
Banking 245, 446, 463
Basic 167, 344, 352, 381, 585
– -Compiler 381
– -Programme 396
Batch-File 470

Batterie 138
Baudot-Code 241
Baudrate 114
BCD-Code 20
BDOS 445, 461
Befehls|kürzel 344
-satz 66
BEL 22
Bell Laboratories 501
– -Standard 275
Betriebssystem 444
Big-Mac-Assembler 330
Bild|röhre 198
-schirm 126
-schirmtext 289
Binär 15
-baum 401, 439
Binäre Logik 15
Binäres Einfügen 587
Biorhythmus 579
BIOS 445, 494
Bipolar 149
Bit 16, 27
Bit/s 274
BNPF 241
Boolesche Algebra 151
Booten 446, 519
Bottom-Up 430
Break-Code 339
-point 316
Breitband-Netzwerk 294
BS 22
Btx 289
Bubblesort 588
Buffer 248
Bus 11, 179
Byte 18

C

C (Sprache) 428, 501
C-64 319, 385
Cache-Speicher 66, 463
CAS 92
CCITT-Nr.-5-Code 23
CCP 445
CCPM 466
Centronics-Schnittstelle 201
Character 21
Checksum 242
Chips 46
CMOS 58, 150
– -Einchip-Mikrocomputer 31
– -Mikroprozessoren 29
– -Schaltungen 158
Cobol 427
Codesegment 335
Codierung 22
COM-Schnittstelle 500
– -File 337
Commodore-Basic 385
– -DOS 518
– -Format 233
Compiler 350, 381
Concurrent-CP/M 466
CONFIG.SYS 491
Context Switch 38
CP/M 245, 445
– -68k 467
– -86 466
– -Assembler 340
– -Plus (CP/M 3) 462
CPU 11, 243
CR 22
CRC 225, 285
CSMA 294

CTC 105
CTRL-Taste 194, 450, 570
Cursor 22
-tasten 490
Cyclic Redundancy Check 225

D

D-Flipflop 100
D/A-Umsetzer 116
Daisy-Chain 188
DART 115
Dateiausgabe 430
Daten|bank 285, 396, 437
-bus 12, 48, 179
-endgerät 206
-felder 225
-pakete 280
-richtungs-Register 102
-separator 221
-strukturen 437
-übertragung 273
Datex-P 280
Datum 136, 579
DCE 207
DDT 451
DEBUG 484
DEE 207
Dehnungsmeßstreifen 171
Deklarationsteil 415
DEL 22
Deleted Data Address Mark 225
Demodulation 276
Dezimalzahlen 20
Diablo-Standard 263
Diagnoseprogramm 399
Diode 147
Directory 469
Disk-Organisation 533
Diskette 216
Diskettenlaufwerk-Schnittstelle 214
Display-Modus 500
Distribution 547
DLAB 500
Dokumentation 545
Doppelflanken-Umsetzer 121
DOS 3.3 517
Dotierung 159
Drain 150
Drucker 254
-steuerung 337
Drucksensor 169
Dual-In-Line-Gehäuse 46
Dual-Slope-Verfahren 121
Duale Zahlen 15
DÜE 207
Dummy 360
Duplex 274, 287, 291, 294
Durchkontaktierung 176
Dynamische Speicher 91

E

E12-Reihe 141, 144
EAN-Strichcode 236
EAROM 99
EBCDIC 25
ECB-System 182
Echtzeituhr 135
ECL-Technik 91
ECMA-70 227
ED (CP/M-Editor) 340
Editieren 354, 376

Editor 340, 480, 482, 571
EDLIN 482
EEPROM 99
Effektive Adresse 333
Einchip-Computer 29
Eingabeschaltung 101
Einplatinen-Computer 13, 246, 251, 543
Einschwingzeit 118
Einzelblatt-Zuführung 261
Elektrische Eigenschaften 55
Elektrolyt 143
Elektronen 147
Elektrosensitiv-Drucker 255
Elko 143, 166
EMUF 251
Emulation 549
Endlosformulare 260
Entscheidung 429
Entwicklungsphasen 561
Entwurf 563
EOF 430, 435
EPROM 96, 245
Epson-kompatibel 258, 337
Ergonomie 540
ESC-Sequenzen 258, 263
ETX-ACK-Prozedur 208
Eumel 514
Europakarte 182
EXE-File 335
EXOR 152
Expertensystem 399
Externe Kommandos 472

F

Fach|ausdrücke 545
-händler 566
-zeitschriften 567
Fädel-Verdrahtung 173
Fahrkarten-Automat 14
Farb|code 141
-norm 125
-träger 125
Fatal Error 499
FCB 448, 498
FDC 225
Fehler abfangen 499
-erkennung 225
Feld|effekt-Transistor 150
-variable 349
Fernsehnormen 125
Ferritkern 145
Festplatte 227
Festwertspeicher 95
FET 150
Feuchtigkeitssensor 171
FIG-Forth 410
File-Control-Block 448, 498
Film 175
Filter 269, 471
Flex 514
Flipflop 91, 154
Floppy-Controller 132
– -Disk-Aufzeichnung 216, 444
– -Laufwerk 214
Fluoreszenz-Anzeige 196
Flußdiagramm 309, 429
Flüssigkristall 149, 197
FM 221, 273
Folientastatur 192
Forth 402
– -II 404
Fortran 426
Fotolack 175
Frame 283
Frequenzmodulation 221, 273
Funktionen 576, 593
Funktionstasten 484, 490

G

Gallium-Arsenid 147
Garbage Collection 360
Gasdruck-Sensor 169
Gatter 152
GCR 223
Gedruckte Schaltung 175
Gehäuse 46, 177, 189
Generatorpolynom 225, 285
Gerade 131
Germanium 147
Geschwindigkeit 350, 540
Gitter 198
Gleichrichter 147, 166
Grafik 131, 264, 290, 590
– -Algorithmen 590
– -Prozessor 131
– -Tablett 301
Grafische Darstellung 429
Graves-Forth 404
Grobkonzept 430, 562
Gruppencodierung 223
GW-Basic 376

H

Halb|duplex 291, 294
-leiter 147
-leiterspeicher 90
Hall-Generator 193
Hand|buch 545
-shake 104, 208
Hard Disk 227
Hash-Puffer 463
HDLC 283
Header 233
Heapsort 589
Heißleiter 171
Heizungssteuerung 14
Hersteller 597
Hex-Code 22, 194, 309
– -Format 242
Hexadezimal 15
Hierarchie 577
Hierarchische Struktur 469
High 16
Hilfskanal 275
Hochspannung 199
Hoff, Ted 26
Höhere Programmiersprachen 340
Hohlnieten 177
Horizontalfrequenz 125

I J

I/O-Adressen (MS-DOS) 499
– -Byte 447
– -Port 100
– -Umleitung 470
IBM-3740-Format 227
– -Basic 376
– -PC 468, 488
– -Zeichensatz 489
ICs 151
ID-Adreßmarke 225
IEC-Bus 210
IEEE-488-Bus 210
IL-Bus 212
Imaginärteil 585
Immediate 67
Implizit 67
Index-Adreßmarke 227
-loch 218

Indirekt 67
Indiziert 67
Induktivität 144
Infobox 288
Institutionen 606
Instruction Queue 65
INT 494
Integrierte Schaltung 151
- Software 543
Intel-Hex-Format 243, 449
Inter|face 13
-face-Loop 212
-leaving 219
-preter 344, 415
-rupt 135, 494, 500
Inverter 152
IORQ 103
iRMX 515
IRQ 54, 340
ISBN 581
ISO-7-Bit-Code 21, 501

Joystick 300

K

Kansas-City-Standard 230
Kapazitive Tastatur 193
Kapital 582
Kartei 437
Kassetten-Aufzeichnung 230
-recorder-Anschluß 234
Katode 147, 198
Kaufvertrag 550
KByte 39
Kennlinie 148
Keramik-Resonator 145
Keyboard-Encoder 136
Kilby, Jack 26
Kildall, Gary 445
Knüppel 301
Kohleschicht 141
Kommando-Eingabe 544
- -Interpreter 445, 449
Kompatibilität 548
Komplexe Arithmetik 585
Kondensator 142
Kopierschutz 554
Korrelationskoeffizienten 582
Kosten 546
Kreise 593
Kupferlackdraht 173

L

LANs 296
Laserdrucker 261
Laufwerks-Schnittstelle 214
Laufzeit-Paket 351
Layout 175
LCD 149, 197
Least Significant Bit 18
LED 148
- -Display 195
Leibniz 25
Leiterplatte 175
Leitungsimpedanz 213
Leuchtdiode 148
LF 22
Licht|griffel 301
-schranke 219
Line-by-Line-Assembler 484
Lineares Einfügen 587
Linearität 124
Linien 592

Link 345
Linker 337
Lisa-Assembler 325
Lizenz 568
Lochrasterplatte 173
Lochstreifen 240
Lockout 138
Logarithmus 577
Logik 151
– -Symbole 153
Logo 428
Lokale Netzwerke 293
– Variable 423
Lötkolben 173
Lottozahlen 582
Low 16
LS-TTL 155, 160
LSB 18
Lüfter 190

M

M2FM-Aufzeichnung 223
Magnetfeld-Messung 172
Mail|box-Dienste 285
-merge 574
Make-Code 339
Makro-Assembler 330, 333
Marketing 565
Maschinensprache 307
Maskenprogrammierung 96
MASM-86 333
Matrix-Drucker 254
Maus 296
MBasic 352, 430
MByte 39
Mehrbenutzer-System 466
Memory Management 501
Menü 544
Meßgenauigkeit 125
Metallpapier 254
MFM-Aufzeichnung 222
Mikrocomputer 11
Mikrofon 304
Mikroprozessor 11, 25
MMU 501
Mnemonics 67
Modem 273
Modul 564
Modula 420
Modulationsart 273
Monitorprogramm 307
Monoflop 158
Monotonie 124
MOS 91
– -Technology-Format 242
MOSFET 158
Most Significant Bit 18
Motorola-S-Format 242
MP/M 465
MS-DOS 468
MSB 18
MSX 515
Multi-Emitter-Transistor 156
-bus-II 184
-layer 178
-plexen 47, 304
-user-System 466, 501

N

N-Kanal-MOSFET 158
N-Key-Rollover 138
Nachindiziert 67
Nachleuchtdauer 303

Nadeldrucker 257
NAND-Gatter 152
Nassi-Shneiderman-Diagramm 433
NDR-Klein-Computer 248
Negative Logik 57
Netz|filter 165
-knoten 280
-teil 164
-werke 296
Newell Noop 294
Nibble 18, 38
NiCd-Akku 138
Nichtlinearität 125
NMOS 58
– -Einchip-Mikrocomputer 30
– -Mikroprozessoren 28
NOR-Gatter 152
No Return To Zero 231
NPN-Transistor 149
NRZ 231
NTSC-Norm 125
NUI 282
NUL 22
Null|durchgangs-Analyse 303
-kraftsockel 97
-modem 206
Numerische Variablen 346

O

Oasis 515
Objektcode 324
Offsetdruck 176
Ohm 140
Oktale Zahlen 19
Open Collector 156
Operationscodes 66
Operatoren 364, 375
Optische Maus 298
Optokoppler 209
Originate 279
Orthogonal 67
OS-9 515
Oszillator 123
Overhead 351

P

P-Code 415, 516
P-Kanal-MOSFET 158
PAD 281
Paketvermittlung 280
PAL 125
Parallel-Schnittstelle 201
Paritätsprüfung 206, 246
Parity-Bit 206
Pascal 415, 436
Paßwort 557
PC-DOS 468
PCOS 518
PDP-7/11 501
PE-Aufzeichnung 231
Pegel 56
Peripherie-Bausteine 100
-geräte 14
Pfad 470
Phase Locked Loop 221, 231
Phasen-Codierung 231
-modulation 273
Pick 515
Pin-Bezeichnungen 48
PIO 103
PIP 453
Pipelining 65

Piping 471
PL/I 426
Platine 175
PLL 221, 231
Plotter 264
PM 273
Polynom 225
Portabilität 548
Portbausteine 100
Positive Logik 57
Postmodem 277
Potentialfreiheit 213
Pressemitteilungen 567
Primzahlen 578
Prioritäts-Steuerung 185
Prodos 518
Programmänderungen 552
Programmierspannung 99
Programmzähler 36
Prologue 518
PROM 96
Prototypen 173
Prüfbit 206
Prüfsumme 242
Pseudobefehle 335
Public Key 558
Punkt|kommandos 574
-raster 127

Q

QDOS 468
Quarz 115, 145
Quell|datei 471
-text 324
Quick Disk 219
-basic-Compiler 394
-sort 589

R

R-2R-Netzwerk 118
R/W-Leitung 12
RAM 11, 90, 243
RAS 92
Rauschgenerator 270
Redundanz 241
Reflex-Lichtschranke 235
Refresh 62, 92
Register 36
Reihenentwicklung 577
Rekursion 413
Relevation 515
Resonanzfrequenz 145
Return 22
RGB-Eingang 127
Richtungserkennung 298
Ring-Netzwerk 294
-puffer 501
RIOT 104
RM-65-Bus 183
Rollen-Berechnung 579
Rollkugel 298
ROM 11, 95, 243
ROM-BIOS 339, 494
Root-Directory 469
RPNL 427
RS-232C 203
Run Time Package 351
Rundungsfehler 347
RWTS-Routine 528
RXD 208

S

S-100-Bus 179
S-Records 242
Sägezahn-Verfahren 120
Sample-and-Hold 123
SBC 13, 251
SC-Assembler 328
Scan-Code 339, 490
SCART-Stecker 200
Schaltnetzteil 164
Scheinwiderstand 142
Schieberegister 106
Schleifen 432
– -Struktur 293
Schlüssel 556
Schmitt-Trigger 156, 304
Schottky 155
Schreib-Lese-Kopf 219
-schutz 219
-weisen 545
Schrittmotor 219
Schwellspannung 147
Schwingkreis 145
Scratchpad Memory 40
Secam-Norm 125
Sedezimal 18
Segment 39, 335
Sektoren 218
Sensoren 168
Serielle Ein-/Ausgabe 112
– Schnittstelle 116, 203, 277, 500
Serieller Bus 294
Shakersort 588
Shapes 370
Shell 502, 510
Shift-Taste 490
Shugart-Schnittstelle 214
Signalprozessor 305
Silizium 147
Single Density 221
SIO 115
Slot 188, 311
Softwarekompatibel 32
Sortieren 587
SOS 518
Source 150, 286, 324
– -Listing 337
SP-Netzwerk 295
Space 21
Spannungs-Vervielfachung 166
-regler 162
-stabilisation 148
Speicher|chips 90
-matrix 92
-verwaltung 463
-zugriff 59
Sperrspannung 147
Spooler 466
Sprach-Spektrum 304
-ausgabe 268
-erkennung 303
-synthese 268
Sprecher-Erkennung 303
Spule 144
Spur 218
Stachelwalze 261
Stack 37, 499
– -Segment 335
-pointer 37
Standardabweichung 582
Startbit 206, 275
Statische Speicher 90
Statistik 583
Status|leitungen 180
-register 36
Steckkarte 178
Stern-Struktur 293
Steuerzeichen 21, 194, 259, 262
Stopbit 206, 275
Stör|impulse 164
-strahlung 164, 190

Streamer 228
Streuung 582
Strichcode 235
– -Leser 168, 299
String-Routinen 585
Strings 347, 438
Strobe 138, 202
Strom|schleife (20 mA) 208
-versorgung 162
Struktogramm 309, 432
Subdirectory 469
Suche 396
Sukzessive Approximation 122
Switch 494
Symbole 429, 435
SYN-Zeichen 232
Syntax-Diagramm 435

T

Takt|frequenz 55
-oszillator 243
Tantal 143
TASC 381
Tasks 466
Tastatur 191
– -Encoder 136
– -Scan-Code 490
TEDAS 287
Telebox 291
Telefon-Modem 273
-netz 277
Telex-Code 24
Temperaturmessung 170
Terminal 280, 287
– -Treiber 492
Text|dateien 531, 538
-editor 340, 480, 482, 571
Thermo-Transfer-Drucker 256
Thermodrucker 255
Thyristor 151
Tiere raten 399, 437
Timer 100
Timesharing 465, 501
Timing 59
Tintenstrahldrucker 261
Tischcomputer 14
Token 344
Tolerante Suche 396
Tonfrequenzen 274
Tonsignal 200
Top-Down 430
TOS 519
TPA 446
Trägerfolie 218
Traktor 261
Transformator 144, 213
Triac 151
Trommelplotter 265
TRSDOS 518
TTL 151, 155
TTY-Schnittstelle 208
Turbo-Pascal 351, 425
TXD 208
Typenkorb-Drucker 261
Typenrad-Drucker 261

U

UART 115
UCSD-P-System 415, 516
UCSD-Pascal 424
UHF-Modulator 126
Uhr 135
Umgekehrte Polnische Notation 403
Umlaute 23, 194, 501
Umsatz-Kalkulation 543
UND-Gatter 152
Unix 501

Unter|längen 127
-programme 37, 431
-verzeichnisse 469
UPN 403
Urheberrecht 551
Urladeprogramm 308
US-ASCII 23
US-Modems 275

V

V.21-Modem 274
V.23-Modem 275
V.24-Schnittstelle 116, 203, 277
Variablen 346
Varianz 582
Verbände 606
Verdrahtung 173
Verkettete Liste 438
Verknüpfung 152
Veröffentlichungen 569
Versa-DOS 516
Versand 566
Verschachtelung 432
Verschlüsseln 559
Vertikalfrequenz 125
VIA 106
Video-Bandbreite 129
– -Controller 125, 500
– -Eingang 199
– -RAM 130
-signal 199
Virtuelle Adressierung 501
– CPU 415
VME-Bus 187

W

Wait-Zyklen 59
Wärme 47, 189
Watchdog-Schaltung 246
Werbung 566
Widerstand 140, 301
Wildcard 451
Wire-Wrap 173
Wordstar 570
Workspace 38
Wortbreite 27

X Z

X.25-Protokoll 280, 282, 283
X.28 283
X.75 283
Xenix 428
XON/XOFF 208

Z-Dioden 148
Zahlendarstellung 15
Zähler 155
Zeichensatz 471, 487
Zeiger 438
Zeilen|sprung 126
-editor 340, 476
Zeitverhalten 58, 61
Zero Page 67, 310
Ziffernblock 191
Zinn 174
Zinseszins 582
Zufallszahlen 582
Zugriffszeit 59
Zuse, Konrad 26
Zweidrahtleitung 213
ZZF 279

Weitere Franzis-Computer-Fachbücher

Betriebssystem CP/M

Vom Monitorprogramm zum Mehrbenutzersystem. Von Jürgen **Plate.** 2., neu bearbeitete und erweiterte Auflage. 422 S., 48 Abb., Lwstr. kart. DM 58.—
ISBN 3-7723-7522-7

Der Band bietet die wohl benutzerfreundlichste und umfassendste Beschreibung des Betriebssystems CP/M.
Stück um Stück werden die Mikrocomputer-Betriebssysteme überhaupt, CP/M als Hauptsache, beschrieben. Der Leser wird vom einfachen Monitorprogramm über das weit verbreitete Betriebssystem CP/M zu den Multiuser- und Multitasking-Betriebssystemen geführt. Der Autor geht sehr in die Tiefe und ins Detail, so daß man hier ohne Übertreibung von einem CP/M-Handbuch sprechen kann.

Mikrocomputer selbstgebaut und programmiert

Vom Bauelement zum fertigen Z-80-Computer. Von Rolf-Dieter **Klein.** 2., neu bearbeitete und erweiterte Auflage. 419 Seiten mit 370 Abbildungen. Lwstr-geb. DM 38.–
ISBN 3-7723-7162-0

Einen Mikrocomputer selbst zu bauen und zu programmieren, das ist das Ziel dieses Buches. Und das Ziel wird erreicht. Wie? Durch ein neuartiges Experimentiersystem. Die Platinen dazu sind im Handel erhältlich oder können selbst hergestellt werden. Durch ein modulares Konzept ergeben sich zahlreiche Kombinationsmöglichkeiten. Der Softwareteil des Bandes bietet für alle im Rechner verwendete Programme ausführliche Ausdrucke.

Hard- und Software für den Epson HX-20

Heiße Applikationen für den attraktiven Aktentaschen-Computer. Von Jürgen **Plate** und Wolfram **Lechner.** 142 Seiten, 19 Abbildungen. Lwstr-geb. DM 38.–
ISBN 3-7723-8041-7

Wenn Sie wissen wollen, was Ihr HX-20 noch zusätzlich leisten kann, dann greifen Sie zu diesem Band. Wo das Original-Handbuch aufhört, dort fangen die Autoren an.
Fünf Punkte sind es, die in die Augen stechen:
1. Zusätzliche Kenntnisse über Hardware und Basic-Interpreter (z. B. PEEK und POKE) bilden die Basis für zusätzliche Applikationen.
2. Über dreißig neue, sorgfältig erprobte Anwendungen aus der Praxis brauchen nur abgeschrieben werden.
3. Hier öffnet sich das Tor zur Maschinensprache des HX-20, mit der bestimmte Programme eben wesentlich schneller sind (Assembler, Bioassembler).
4. Interessante Hardwareerweiterungen machen den Hand-Held wesentlich leistungsfähiger (Speichererweiterung, Schnittstelle für Messen, Steuern und Regeln).
5. Für die routinierten Anwender ein Glossar mit Speicherbelegung und Systemroutinen.

Computer-Grafik: Einführung — Algorithmen — Programmentwicklung

Ein Arbeitsbuch mit zahlreichen Programmen und Hardware-Applikationen für Plotter und Bildschirm. Von Jürgen **Plate.** 415 Seiten mit 252 Abbildungen und 5 Tabellen. Lwstr.-kart. DM 68.—
ISBN 3-7723-8281-9

Jeder Anwender dieses Arbeitsbuches findet hier eine gründliche und praktische Anleitung, Computergrafiken zu erstellen, mit ihnen zu arbeiten und sie richtig auszuwerten.
Der Band hebt sich von den anderen Büchern über Computer-Grafik dadurch ab, daß er sich nicht auf eine Bildschirmdarstellung beschränkt, sondern ebenfalls umfassend die Plotteransteuerung behandelt.
Der Benutzer dieses Bandes findet hier genau die richtigen Werkzeuge für seine erfolgreiche Grafik-Arbeit am Computer. Sei es für Hobby oder Beruf.

Erfolgreicher mit CBM arbeiten

Für alle CBM-Anwender eine verständliche Einführung in die Maschinensprache. Von Dipl.-Ing. Franz **Wunderlich.** 248 Seiten, 11 Abbildungen. Lwstr-geb. DM 42.—
ISBN 3-7723-7052-7

CBM-Anwender mit Basic-Erfahrung holen mit diesem Buch mehr aus ihrem Computer heraus. Sie können nämlich mit dem Programmieren in der Maschinensprache beginnen. Zunächst wird ihnen beigebracht, wie ein 6502 programmiert wird. Zahlensysteme, Speicherkozepte, Adressierung und Befehlssätze werden behandelt und gewinnen Klarheit. Im Hauptteil wird speziell die geräteabhängige Software der Commodore-Serie CBM abgehandelt.
Natürlich bringt der Autor auch fertige Programme und viele CBM-spezielle Anwenderbeispiele im Anhang.

Pascal: Einführung – Programmentwicklung – Strukturen

Ein Arbeitsbuch mit zahlreichen Programmen, Übungen und Aufgaben. Von Jürgen **Plate** und Paul **Wittstock.** 3., neu bearbeitete und erweiterte Auflage. 426 Seiten, 184 Abbildungen. Lwstr-kart. DM 58.–
ISBN 3-7723-6903-0

Schritt für Schritt wird der Leser in das Programmieren mit Pascal eingeführt. Die Autoren haben sich in die Ahnungslosigkeit des Anfängers hineinversetzt. Sie bringen ihm das besondere Denken des routinierten Programmierers bei. Das Verblüffende dabei ist, sie kommen mit einer einfachen klaren Sprache aus, verabscheuen das EDV-Chinesisch, setzen nichts voraus. Wer sich an dieses Buch heranmacht, meint, es gäbe nichts Einfacheres als Pascal.

Preisänderungen und Liefermöglichkeit vorbehalten

Franzis-Verlag, München

Weitere Franzis-Computer-Fachbücher

Z-80-Applikationen

Eine Sammlung von gut dokumentierten Programmen, die universell anwendbar sind. Von M. und R.-D. **Klein.** 3., verbesserte und erweiterte Auflage. 164 Seiten, 114 Abbildungen.
Lwstr-geb. DM 38.–
ISBN 3-7723-6673-2

Das Applikationsbuch ist ein Schritt in die Richtung „Lösen von Standardproblemen".
Bewußt legt der Autor den Schwerpunkt auf die sofortige Anwendbarkeit der dargestellten Methoden und Programme. Er bietet Standardlösungen z. B. für die Ein-/Ausgabe über eine serielle Schnittstelle oder Interruptschaltung, führt Programme zur Meßwertverarbeitung an und zeigt, wie man sich nützliche Arbeitshilfen schaffen kann.

Erfolgreicher mit dem VC 64 arbeiten

Eine verständliche Einführung für alle VC 64-Anwender in die Maschinensprache. Von Dipl.-Ing. Franz **Wunderlich.** 192 Seiten, 3 Abbildungen.
Lwstr-geb. DM 38.–
ISBN 3-7723-7781-5

Wer einen VC 64 sein eigen nennt, bekommt hier eine Unterweisung, die er sich eigentlich von Anfang an gewünscht hat. Ein äußerst präziser und differenzierter Lehrgang zeigt ihm das Programmieren in der Maschinensprache. Die Eigenart seiner Maschine wird ihm dabei in allen Dimensionen deutlich, so daß das Schreiben eines guten Programmes in der Maschinensprache von vornherein erfolgreich ist.
Es sei hier stichwortartig auf ein paar Einzelheiten hingewiesen: Grundsätzliches zur Maschinensprache der 65xx Serie. Basic-Interpreter und Betriebssystem auf der Maschinenebene betrachtet. Liste der ROM-Routinen mit hex- und dezimalen Adressen. Tabellen zur Umrechnung bei der Maschinenprogrammierung.
Dieses Buch begeistert alle VC 64-Anwender, auch die Anfänger. Dem Routinier dient es als zuverlässiges Nachschlagewerk.

Microsoft-Basic: Konzepte, Algorithmen, Datenstrukturen

Eine Einführung und ein Arbeitsbuch mit der Gesamtkonzeption. Von Luidger **Röckrath.**
224 Seiten, 96 Abbildungen.
Lwstr-geb. DM 48.–
ISBN 3-7723-8011-5

Es wird der Aufbau und die Verwaltung des Programmtextes und der Variablentabelle erläutert ebenso die Operationen, die mit den Daten der verschiedenen Rechnertypen durchgeführt werden können. Die Organisation der Ein-/Ausgabe mit der Betriebssystemanbindung wird ausführlich beschrieben. Die komplexen Möglichkeiten im Microsoft-Basic zur Bildung von Ausdrücken werden zur Verfügung gestellt.
Fazit: Was ein Programmierer mit Microsoft-Basic organisieren kann, ist hier zusammengefaßt und ausführlich beschrieben. So bildet sich bei dem Leser ein Verständnis für Microsoft-Basic, das von dem fruchtlosen Block-Box-Denken wegführt.

IEC-Bus

Die Funktionsweise des IEC-Bus und seine Anwendung in Geräten und Systemen. Von Dr. Anton **Piotrowski.** 2., verbesserte und erweiterte Auflage. 312 Seiten mit 128 Abbildungen und 105 Tabellen.
Lwstr-geb. DM 48.–
ISBN 3-7723-6952-9

Hier liegt eine fachgerechte Darstellung der Funktionsweise des IEC-Bus und dessen Anwendung in den Geräten und den Systemen vor. Diese gliedert sich in drei Teile: 1. Theoretische Grundlagen der Funktion des IEC-Bus mit ausführlicher Beschreibung der Nachrichten- und der Zustandsdiagramme. 2. Beschreibung der IEC-Interface-Bausteine und deren Anwendungen bei der Realisierung von IEC-Schnittstellen mit Mikroprozessoren. 3. Gerätetechnik, Aufbau und Programmierung von IEC-Bus-Systemen, inklusiv der Fehleranalyse.

Ausgewählte und in sich abgeschlossene praktische Anwendungsbeispiele in Hard- und Software erleichtern dem Techniker und Informatiker die tägliche Arbeit.

Das große CPC-Arbeitsbuch

Eine aktuelle und praktische Anleitung, das Optimale aus den Schneider-Computern herauszuholen — für CPC 464, 664 und 6128. Von Lothar **Miedel** und Martin **Kotulla.** 453 Seiten mit 30 Abbildungen. Lwstr.-kart. DM 68.—
ISBN 3-7723-8421-8

Das große CPC-Arbeitsbuch fängt dort an, wo die Handbücher zu den Schneider CPCs aufhören. Das große CPC-Arbeitsbuch ist das Bindeglied zwischen Handbuch und praktischem Einsatz. Das große CPC-Arbeitsbuch bietet viele Lösungen für den täglichen Umgang mit dem CPC.
Für Programmierer, die nicht nur Software von der Stange auf ihrem CPC laufen lassen wollen, ist das große CPC-Arbeitsbuch ein Muß!

Hardware-Erweiterungen für Sharp-Taschencomputer

Applikationen und Software zum Messen und Steuern. Von Burkhard **Kainka.** 111 Seiten mit 28 Abbildungen. Lwstr.-kart. DM 48.—
ISBN 3-7723-8351-3

Das modulare System der Sharp-Taschencomputer läßt sich mit Hilfe dieses Bandes um wichtige und entscheidende Anwendungsmöglichkeiten erweitern. Das Spektrum reicht von einfachen Geräten wie einem Zeitschalter oder Impulszähler, bis hin zu anspruchsvollen größeren Projekten, wie einem Transientenrecorder. Der Autor stellt die notwendigen und einfach aufgebauten Interface-Schaltungen für das Messen und Steuern fertig entwickelt zur Verfügung.

Preisänderungen und Liefermöglichkeit vorbehalten.

Franzis-Verlag, München

FEICHTINGER, H: ARBEIT PP
SB.MIKROCOMP 24909
108,00 94386
10087
N4893980 000
N3772380220